

Anatol GREMALSCHI

Iurie MOCANU

Ion SPINEI

INFORMATICA

Limbajul PASCAL



Manual pentru clasele IX-XI

A. Gremalschi, I. Mocanu, I. Spinei

Informatica

Limbajul PASCAL

Manual pentru clasele IX-XI

Chişinău, Î.E.P. *Ştiinţa*
1999

CZU 004 (075.3)

*Aprobat pentru editare de Consiliul Ministerului Educației
și Științei al Republicii Moldova
(nr. 1/4.1 din 26 august 1998)*

Redactor: *E. Grosu*
Tehnoredactor: *N. Duduciuc*
Corectori: *M. Belenciuc, E. Pistrui*

Machetare computerizată: *Î.E.P. Știința*

Coli editoriale 14,0. Coli de tipar conv. 16,0
Comanda nr 1115

Întreprinderea Editorial-Poligrafică *Știința*
str. Academiei, 3; MD 2028, Chișinău, Republica Moldova
Tel. (3732) 73-96-16; 73-99-83. Fax (3732) 73-96-27

Firma Editorial-Poligrafică *Tipografia Centrală*
str. Florilor, 1, or. Chișinău, MD 2068, Republica Moldova
Departamentul Activități Editoriale,
Poligrafie și Aprovizionare cu Cărți

ISBN 9975-67-103-9

©A. Gremalschi, I. Mocanu,
I. Spinei, 1999

©Coperta: Oleg Beșliu, 1999

Cuprins

Introducere.	5
Capitolul 1. VOCABULARUL ȘI SINTAXA LIMBAJULUI	
1.1. Inițiere în limbajul PASCAL.	6
1.2. Metalimbajul BNF.	8
1.3. Diagrame sintactice.	10
1.4. Alfabetul limbajului.	14
1.5. Vocabularul limbajului.	14
1.6. Simbolurile speciale și cuvintele-cheie.	14
1.7. Identificatori.	16
1.8. Numere.	17
1.9. Șiruri de caractere.	20
1.10. Etichete.	21
1.11. Directive	21
1.12. Separatori.	22
Capitolul 2. TIPURI DE DATE SIMPLE	
2.1. Conceptul de dată.	24
2.2. Tipul de date <i>integer</i>	25
2.3. Tipul de date <i>real</i>	27
2.4. Tipul de date <i>boolean</i>	30
2.5. Tipul de date <i>char</i>	32
2.6. Tipuri de date <i>enumerare</i>	34
2.7. Tipuri de date <i>subdomeniu</i>	37
2.8. Generalități despre tipurile ordinale de date.	40
2.9. Definirea tipurilor de date.	44
2.10. Declarații de variabile.	49
2.11. Definiții de constante.	51
Capitolul 3. INSTRUCȚIUNI	
3.1. Conceptul de acțiune.	56
3.2. Expresii.	57
3.3. Evaluarea expresiilor.	62
3.4. Tipul expresiilor PASCAL.	64
3.5. Instrucțiunea de atribuire.	67
3.6. Instrucțiunea <i>apel de procedură</i>	69
3.7. Afișarea informației alfanumerice.	71
3.8. Citirea datelor de la tastatură.	74
3.9. Instrucțiunea de efect nul.	77
3.10. Instrucțiunea <i>if</i>	77
3.11. Instrucțiunea <i>case</i>	81
3.12. Instrucțiunea <i>for</i>	84
3.13. Instrucțiunea compusă.	88
3.14. Instrucțiunea <i>while</i>	91
3.15. Instrucțiunea <i>repeat</i>	95
3.16. Instrucțiunea <i>goto</i>	98
3.17. Generalități despre structura unui program PASCAL.	102
Capitolul 4. TIPURI DE DATE STRUCTURATE	
4.1. Tipuri de date <i>tablou</i> (<i>array</i>).	105
4.2. Tipuri de date <i>șir de caractere</i>	112

4.3. Tipuri de date <i>articol</i> (record).	116
4.4. Instrucțiunea <i>with</i> .	120
4.5. Tipuri de date <i>mulțime</i> (set).	124
4.6. Generalități despre fișiere.	128
4.7. Fișiere secvențiale.	132
4.8. Fișiere cu acces direct.	135
4.9. Fișiere <i>text</i> .	138

Capitolul 5. FUNCȚII ȘI PROCEDURI

5.1. Subprograme.	144
5.2. Funcții.	145
5.3. Proceduri.	149
5.4. Parametri formali <i>funcție</i> / <i>procedură</i> .	153
5.5. Domenii de vizibilitate.	158
5.6. Comunicarea prin variabile globale.	161
5.7. Efecte colaterale.	163
5.8. Recursia.	167
5.9. Declarații anticipate.	170
5.10. Sintaxa declarațiilor și apelurilor de subprograme.	172

Capitolul 6. STRUCTURI DINAMICE DE DATE

6.1. Variabile dinamice. Tipul <i>referință</i> .	176
6.2. Structuri de date.	180
6.3. Liste unidirecționale.	181
6.4. Prelucrarea listelor unidirecționale.	186
6.5. Liste bidirecționale.	193
6.6. Stiva.	197
6.7. Cozi.	202
6.8. Arbori binari.	206
6.9. Parcurgerea arborilor binari.	213
6.10. Arbori binari de căutare.	218
6.11. Arbori de ordinul <i>m</i> .	224
6.12. Tipul de date <i>pointer</i> .	230

Capitolul 7. METODE DE ELABORARE A PRODUSELOR PROGRAM

7.1. Programarea modulară.	238
7.2. Testarea și depanarea programelor.	245
7.3. Elemente de programare structurată.	249

<i>Anexa 1.</i> Vocabularul limbajului PASCAL.	252
<i>Anexa 2.</i> Sintaxa limbajului PASCAL.	252
Bibliografie.	256

INTRODUCERE

Limbajul de programare PASCAL a fost elaborat în anul 1970 de profesorul Niklaus Wirth de la Universitatea Tehnică Zürich (Elveția). Numit astfel în cinstea matematicianului și filosofului francez Blaise Pascal, limbajul a cunoscut o răspîndire mondială și a fost implementat practic pe toate categoriile de calculatoare: supercalculatoare, calculatoarele mari și calculatoarele personale. Pe parcursul anilor elementele limbajului standard PASCAL au fost îmbogățite substanțial, creîndu-se astfel versiunile Turbo PASCAL 6.0, Turbo PASCAL 7.0, Borland PASCAL și chiar limbaje noi, de exemplu, MODULA-2, DELPHI ș.a.

Manualul de față a fost elaborat în conformitate cu Programa de informatică, adoptată de Ministerul Educației și Științei al Republicii Moldova și are drept scop însușirea de către cititor a cunoștințelor necesare pentru elaborarea și depanarea programelor PASCAL.

Capitolul 1 include expunerea unor cunoștințe fundamentale din teoria limbajelor de programare: metalimbajul BNF și diagramele sintactice. În continuare este prezentat alfabetul și vocabularul limbajului PASCAL: simbolurile speciale și cuvintele-cheie, identificatorii, numerele, șirurile de caractere, etichetele, directivele și separatorii.

În capitolul 2 sînt expuse tipurile de date întregi, reale, booleene, caracteriale, enumerare și subdomeniu. Tot în acest capitol se studiază mijloacele de definire a tipurilor de date simple, declarațiile de variabile și definițiile de constante.

Capitolul 3 este consacrat conceptului de acțiune și conține descrierea instrucțiunilor simple și instrucțiunilor structurate ale limbajului. O atenție deosebită se acordă studierii sintaxei și metodelor de evaluare a expresiilor PASCAL, modului de organizare a proceselor de calcul cu ajutorul instrucțiunilor structurate. Tot în acest capitol sînt incluse generalități despre structura unui program PASCAL.

Capitolul 4 include un amplu material teoretic și practic referitor la definirea și prelucrarea datelor structurate: tablourilor, șirurilor de caractere, articolelor, mulțimilor și fișierelor. Sînt prezentate metodele de creare și prelucrare a fișierelor: asocierea fișierelor PASCAL cu fișiere externe, scrierea și citirea componentelor unui fișier.

Funcțiile și procedurile limbajului PASCAL se studiază în capitolul 5. Sînt prezentate în detalii mecanismele de comunicare prin parametri-valoare, parametri-variabilă, parametri-funcție/procedură și variabile globale. Tot în acest capitol se studiază subprogramele recursive.

În capitolul 6 se studiază tipul referință, variabilele dinamice și structurile de date create pe baza lor: liste, stive, cozi, arbori. Sînt examinate operațiile uzuale efectuate asupra structurilor dinamice de date și problemele legate de aplicarea iterației și recursiei.

Metodele de elaborare a produselor program sînt expuse în capitolul 7. Sînt prezentate elemente de programare modulară și programare structurată, metode de testare și depanare a programelor.

Materialul este expus în așa mod, încît sintaxa și semantica limbajului PASCAL să fie înțelese și însușite metodic, de la instrucțiuni și tipuri de date simple la structuri dinamice de date, programe, proceduri și funcții. Fiecare porțiune de materie teoretică este urmată de un șir de exemple, exerciții și întrebări de control. Se consideră că cititorul va introduce și va lansa în execuție programele propuse, va răspunde la întrebări și, în caz de necesitate, va elabora programele respective.

Toate programele incluse în manual au fost testate și depănate în mediul de programare Turbo PASCAL 7.0.

Capitolul 1

.....

VOCABULARUL ȘI SINTAXA LIMBAJULUI

1.1. INIȚIERE ÎN LIMBAJUL PASCAL

Vom examina următorul program:

```
1  Program P1;  
2  { Suma numerelor întregi x,y,z }  
3  var x, y, z, s : integer;  
4  begin  
5      writeln('Introduceți numerele întregi x,y,z:');  
6      readln(x, y, z);  
7      s:=x+y+z;  
8      writeln('Suma numerelor introduse:');  
9      writeln(s);  
10 end.
```

Numerele 1, 2, 3, ..., 10 din partea stângă a paginii nu fac parte din programul PASCAL. Ele servesc doar pentru referirea liniilor în explicațiile ce urmează.

Linia 1. Cuvântul **program** este un cuvânt rezervat al limbajului, iar P1 este un cuvânt utilizator. Cuvintele rezervate servesc pentru perfectarea programelor, iar cuvintele utilizator — pentru denumirea variabilelor, subalgoritmilor, programelor, constantelor ș. a.

Linia 2. Este un text explicativ, un comentariu. Comentariul începe cu simbolul „{” și se termină cu „}”. Comentariul nu influențează în nici un fel derularea programului și este destinat exclusiv utilizatorului.

Linia 3. Cuvântul rezervat **var** (*variable* — variabilă) descrie variabilele x, y, z și s, utilizate în program. Cuvântul **integer** (*întreg*) indică tipul variabilelor respective. Prin urmare, x, y, z și s pot avea ca valori numai numere întregi. Linia în studiu formează partea declarativă a programului.

Linia 4. Cuvântul rezervat **begin** (început) indică începutul părții executabile a programului.

Linia 5. Afișarea unui mesaj la dispozitivul-standard de ieșire, în mod obișnuit pe ecran. Cuvântul `writeln` (*write line* — scrie și trece la linie nouă) reprezintă apelul unui subalgoritm-standard, argumentul fiind textul mesajului ce se afișează:

Introduceți numerele întregi x, y, z :

Menționăm că apostrofurile nu fac parte din textul care va fi afișat.

Linia 6. Citirea a trei numere de la dispozitivul-standard de intrare, în mod obișnuit — tastatura. Numerele sînt tastate în aceeași linie și sînt despărțite de unul sau mai multe spații. După tastarea ultimului număr se acționează tasta <ENTER>. Numerele citite sînt depuse în variabilele x, y, z . Cuvântul `readln` (*read line* — citire și trecere la linie nouă) reprezintă apelul unui subalgoritm-standard. Argumentele subalgoritmului sînt numele variabilelor în care sînt memorate numerele întregi introduse.

Linia 7. Instrucțiunea de atribuire. Variabila s primește valoarea $x + y + z$.

Linia 8. Afișarea mesajului

Suma numerelor introduse:

la dispozitivul-standard de ieșire.

Linia 9. Afișarea valorii variabilei s la dispozitivul-standard de ieșire.

Linia 10. Cuvântul rezervat **end** indică sfîrșitul părții executabile, iar punctul — sfîrșitul programului.

Prin urmare, un program în limbajul PASCAL este alcătuit din următoarele componente:

- ✓ antetul, în care se specifică denumirea programului;
- ✓ partea declarativă, în care se descriu variabilele, funcțiile, subalgoritmii etc. folosiți în program;
- ✓ partea executabilă, care include instrucțiunile ce urmează să fie executate într-o anumită ordine de calculator.

Întrebări și exerciții

- ❶ Introduceți și lansați în execuție programul P1.
- ❷ Care sînt părțile componente ale unui program PASCAL?
- ❸ Introduceți și lansați în execuție următorul program:

```
Program P2;  
{ Afișarea constantei predefinite MaxInt }  
begin  
  writeln('MaxInt=', MaxInt);  
end.
```


- ④ Indicați antetul, partea declarativă și partea executabilă ale programului P2. Explicați destinația fiecărei linii a programului în studiu.
- ⑤ Elaborați un program care afișează pe ecran pătratul și cubul numărului întreg introdus de la tastatură.

1.2. METALIMBAJUL BNF

Un limbaj de programare se definește prin sintaxa și semantica lui. E cunoscut faptul că sintaxa este un set de reguli care guvernează alcătuirea propozițiilor, iar semantica este un set de reguli care determină înțelesul, semnificația propozițiilor respective. În cazul limbajelor de programare, echivalentul *propoziției* este *programul*.

Evident, sintaxa oricărui limbaj de programare poate fi descrisă cu ajutorul unui limbaj de comunicare între oameni, de exemplu româna, engleza, franceza etc. S-a considerat însă că o astfel de descriere este voluminoasă și neunivocă. Pentru o descriere concisă și exactă a sintaxei limbajelor de programare s-au elaborat limbajele speciale, denumite *metalimbaje*. Cel mai răspândit metalimbaj este cunoscut sub denumirea de *BNF — Forma Normală a lui Backus*.

Metalimbajul BNF utilizează următoarele simboluri:

simbolurile terminale, adică simbolurile care apar exact la fel și în programele PASCAL;

simbolurile neterminale, care desemnează unitățile (construcțiile) gramaticale ale limbajului.

Simbolurile neterminale se înscriu între semnele „<” și „>”.

De exemplu, cifrele 0, 1, 2, ..., 9, literele A, B, C, ..., Z sînt simboluri terminale, iar <Cifră>, <Literă> sînt simboluri neterminale.

Descrierea sintaxei limbajului PASCAL constă dintr-un set de formule metalingvistice.

Prin **formulă metalingvistică** vom înțelege o construcție formată din două părți, stînga și dreapta, separate prin simbolurile „::=” ce au semnificația de „*egal prin definiție*”. În partea stîngă a formulei se găsește un simbol neterminal.

O formulă metalingvistică permite descrierea, în partea ei dreaptă, a tuturor alternativelor posibile de definire a simbolului neterminal, prin folosirea caracterului „|” cu semnificația „*sau*”.

De exemplu, formula

<Cifră> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

definește unitatea gramaticală <Cifră> ca fiind unul din caracterele (simbolurile terminale) 0, 1, ..., 9.

La fel se interpretează și formula metalingvistică:

$$\begin{aligned} \langle \text{Literă} \rangle := & \quad a \mid b \mid c \mid d \mid e \mid f \mid g \mid \\ & h \mid i \mid j \mid k \mid l \mid m \mid n \mid \\ & o \mid p \mid q \mid r \mid s \mid t \mid u \mid \\ & v \mid w \mid x \mid y \mid z \end{aligned}$$

În partea dreaptă a unei formule metalingvistice pot apărea două și mai multe simboluri consecutive. Situația corespunde operației de **concatenare** (alipire) a lor.

Astfel:

$$\langle Id \rangle ::= \langle Literă \rangle \mid \langle Literă \rangle \langle Cifără \rangle$$

definește construcția gramaticală $\langle Id \rangle$ ca fiind o literă sau o literă urmată de o cifră.

Exemple: a, a1, c3, x4, d, e8.

În unele situații alternativele de definire a unui simbol neterminal se pot repeta de un număr oarecare de ori (chiar de zero ori), fapt ce va fi marcat prin încadrarea lor în acoladele $\{, \}$.

De exemplu, formula

$$\langle \text{Întreg fără semn} \rangle ::= \langle Cifără \rangle \{ \langle Cifără \rangle \}$$

definește simbolul neterminal $\langle \text{Întreg fără semn} \rangle$ ca o secvență nevidă de cifre. Secvențele 0, 0000, 001, 1900, 35910 sînt conform acestei definiții, iar secvența 3a5910 — nu.

Formula

$$\langle \text{Identificator} \rangle ::= \langle Literă \rangle \{ \langle Literă \rangle \mid \langle Cifără \rangle \}$$

are următoarea semnificație: un identificator începe cu o literă; după această literă poate urma o secvență finită de litere sau cifre. Astfel, a, a1, a1b, a23x, a14bxz sînt conforme cu această definiție, dar 2a — nu.

În cazul în care alternativele de definire a unui neterminal sînt opționale (pot lipsi), ele se încadrează în parantezele drepte $[,]$.

De exemplu, formula

$$\langle \text{Factor scală} \rangle ::= [+ \mid -] \langle \text{Întreg fără semn} \rangle$$

definește factorul de scală ca un număr întreg fără semn care poate fi precedat de + sau -. Astfel, 1, +1, -1, 20, +20, -20, +003 sînt conforme cu această definiție, dar 3-5 — nu.

Atragem atenția că simbolurile $[,], \{, \}$ aparțin metalimbajului și nu trebuie confundate cu simbolurile corespunzătoare utilizate în programul PASCAL.

Întrebări și exerciții

- ❶ Explicați termenii *sintaxă* și *semantică*.

- ② Care este destinația unui metalimbaj?
- ③ Cum se definește sintaxa unui limbaj cu ajutorul metalimbajului BNF?
- ④ Sintaxa unui limbaj foarte simplu este descrisă cu ajutorul următoarelor formule metalingvistice:

$\langle \text{Cifră} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

$\langle \text{Semn} \rangle ::= + \mid -$

$\langle \text{Expresie aritmetică} \rangle ::= \langle \text{Număr} \rangle \{ \langle \text{Semn} \rangle \langle \text{Număr} \rangle \}$

Care din expresiile aritmetice ce urmează sînt numere?

0	0+0	0000
1	11100+1	0001
11100	11100-1	-152
00011	931	+351
20013	614	412

Care din expresiile aritmetice ce urmează sînt corecte?

0+1	-13	21+00000
1+0-3	21+-16	39+00001
0+0+4	-21-16	00001-00001
1+1-9	68-13	379-486
6+6+21	42+650	31+12-51+861

- ⑤ Sintaxa unui limbaj de comunicare utilizator-calculator este definită după cum urmează:

$\langle \text{Disc} \rangle ::= A : \mid B : \mid C : \mid D : \mid E :$

$\langle \text{Listă parametri} \rangle ::= \langle \text{Disc} \rangle \{ , \langle \text{Disc} \rangle \}$

$\langle \text{Nume comandă} \rangle ::= \text{Citire} \mid \text{Copiere} \mid \text{Formatare}$

$\langle \text{Comandă} \rangle ::= \langle \text{Nume comandă} \rangle \langle \text{Listă parametri} \rangle$

Indicați comenzile corecte sintactic:

- | | |
|--------------------|--------------------|
| a) Citire | f) Copiere A: B: |
| b) Citire A: | g) Citire D |
| c) Copiere F: | h) Formatare D:,F: |
| d) Copiere A:, | i) Copiere E:,A:, |
| e) Formatare D:,E: | j) Copiere F:,A: |

1.3. DIAGrame SINTACTICE

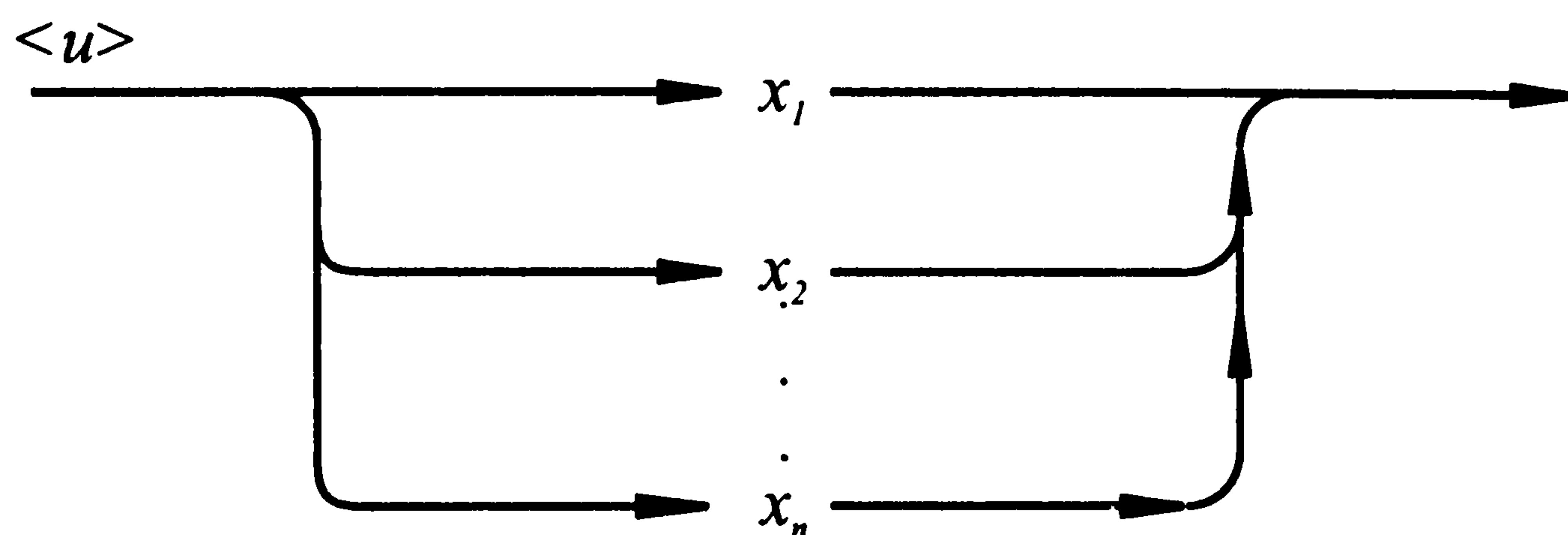
Diagramele sintactice descriu mai clar sintaxa unui limbaj de programare. Reprezentarea prin diagrame poate fi derivată din notația BNF, după cum urmează.

Fiecărui simbol terminal îi corespunde un cerc sau un oval în care se înscrie simbolul respectiv. Simbolurile neterminale se înscriu în dreptunghiuri. Ovalurile și dreptunghiurile se reunesc conform diagramei din *fig. 1.1*.

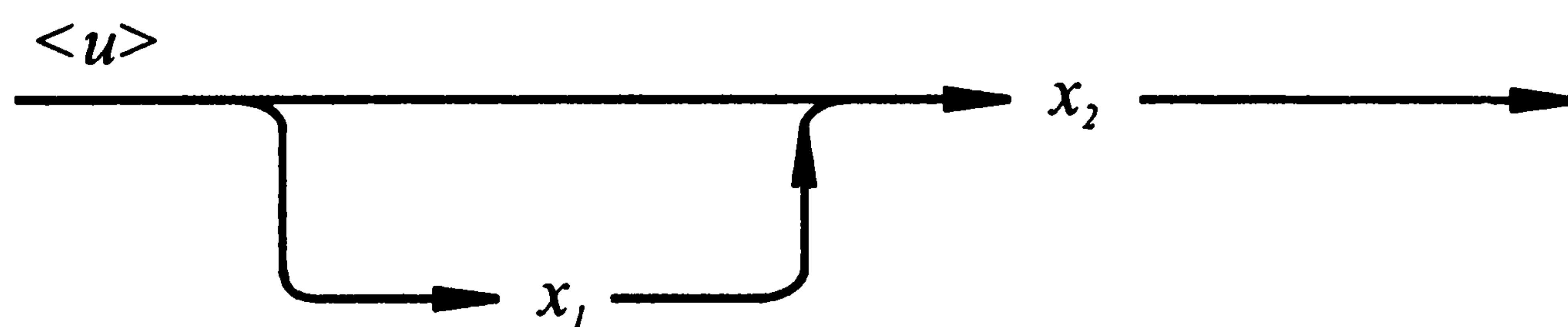
Concatenare: $\langle u \rangle ::= x_1 x_2 \dots x_n$



Alternare: $\langle u \rangle ::= x_1 | x_2 | \dots | x_n$



Prezență opțională: $\langle u \rangle ::= [x_1] x_2$



Repetare: $\langle u \rangle ::= \{x_1\} x_2$

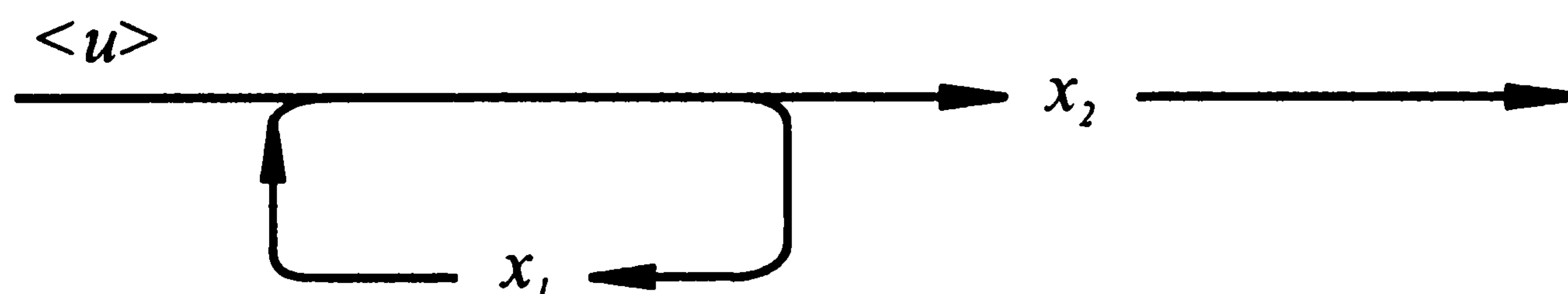


Fig. 1.1. Reprezentarea formulelor BNF prin diagrame sintactice

În fig. 1.2 sînt prezentate diagramele sintactice pentru unitățile gramaticale $\langle \text{Întreg fără semn} \rangle$, $\langle \text{Identificator} \rangle$ și $\langle \text{Factor scală} \rangle$, definite în paragraful precedent. Se observă că fiecărui drum în diagrama sintactică îi corespunde o secvență de simboluri terminale corectă sintactic.

Întrebări și exerciții

- ❶ Care este destinația diagramelor sintactice?
- ❷ Cum se reprezintă simbolurile terminale și simbolurile neterminale pe diagramele sintactice?

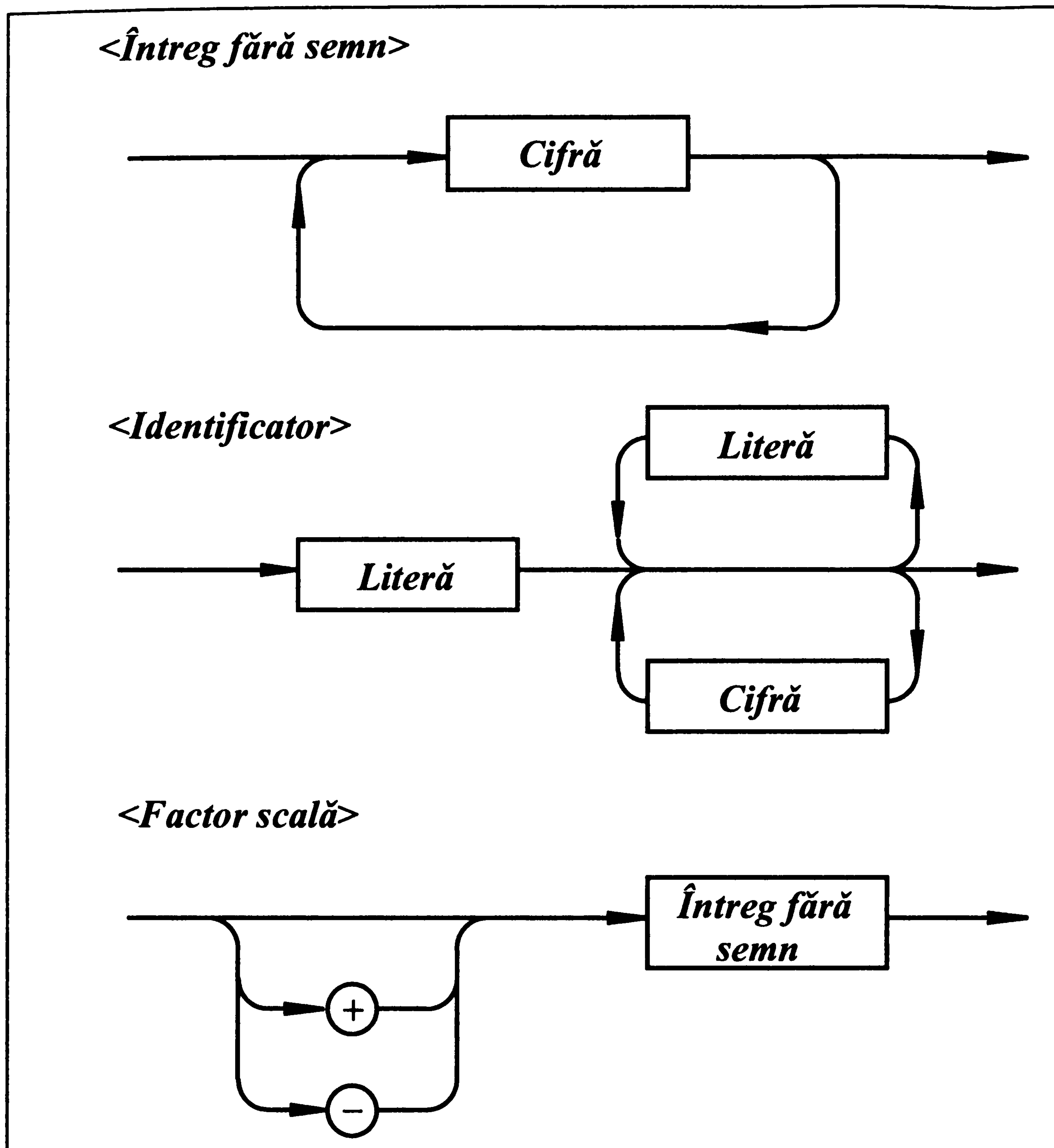


Fig. 1.2. Diagramele sintactice <Întreg fără semn>, <Identificator> și <Factor scală>

- ③ Cum se reprezintă formulele BNF pe diagramele sintactice?
- ④ Reprezentați cu ajutorul diagramei sintactice:

$\langle \text{Cifră} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

$\langle \text{Semn} \rangle ::= + \mid -$

$\langle \text{Expresie aritmetică} \rangle ::= \langle \text{Număr} \rangle \{ \langle \text{Semn} \rangle \langle \text{Număr} \rangle \}$

- ⑤ Reprezentați cu ajutorul diagramei sintactice:

$\langle \text{Disc} \rangle ::= A : \mid B : \mid C : \mid D : \mid E :$

$\langle \text{Listă parametri} \rangle ::= \langle \text{Disc} \rangle \{ , \langle \text{Disc} \rangle \}$

$\langle \text{Nume comandă} \rangle ::= \text{Citire} \mid \text{Copiere} \mid \text{Formatare}$

$\langle \text{Comandă} \rangle ::= \langle \text{Nume comandă} \rangle \langle \text{Listă parametri} \rangle$

- ⑥ În *fig. 1.3* sînt prezentate diagramele sintactice care definesc unitatea gramaticală *<Număr fracționar>*. Determinați care din secvențele ce urmează sînt conforme acestor diagrame:

0.1	.538
+0.1	721.386.
-0.0	-421
9.000	247.532
-538.	+109.000

- ⑦ Scrieți formulele BNF care corespund diagramelor sintactice din *fig. 1.3*.

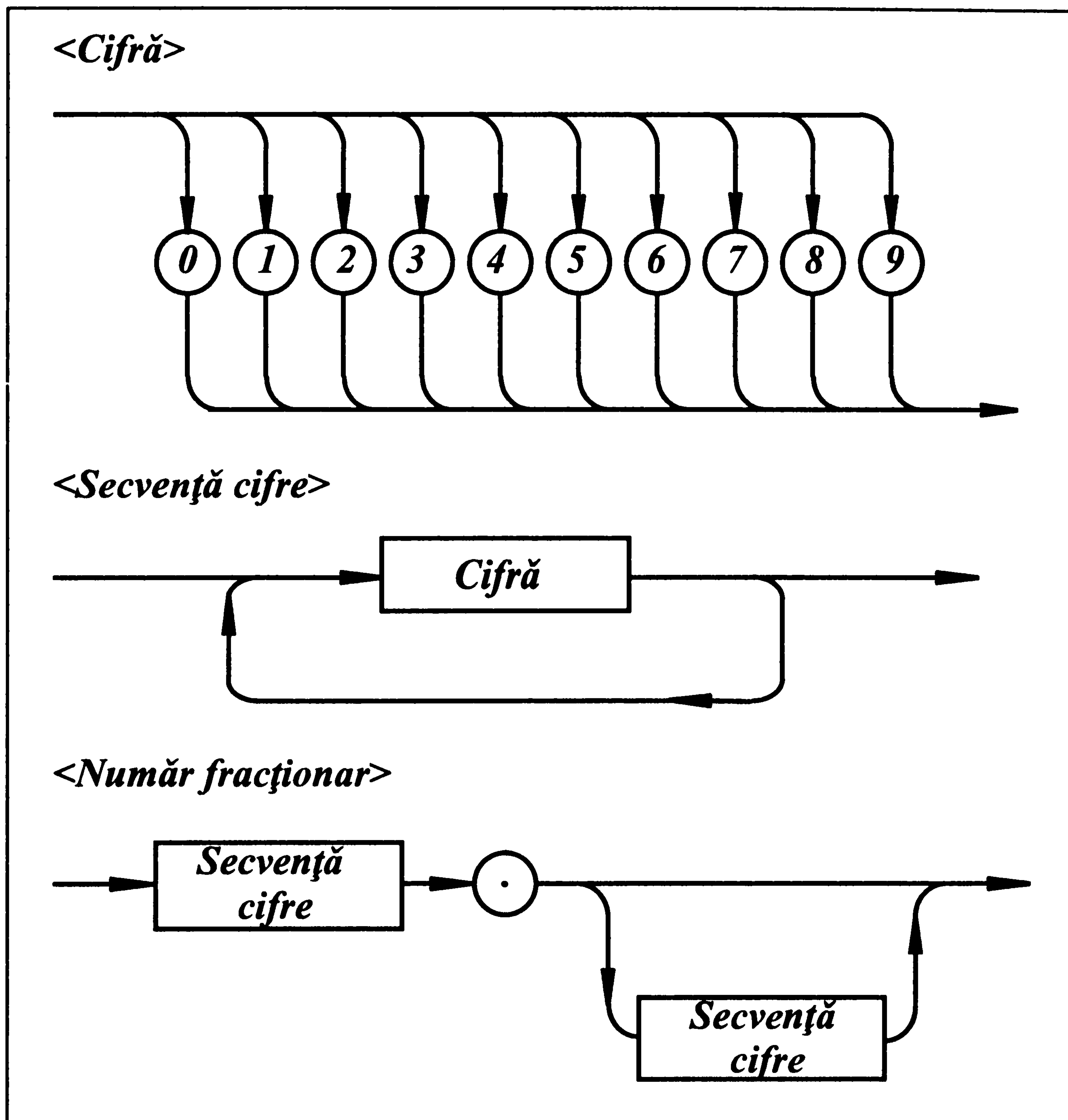


Fig. 1.3. Diagramele sintactice *<Cifră>*, *<Secvență cifre>*, *<Număr fracționar>*

1.4. ALFABETUL LIMBAJULUI

Alfabetul limbajului PASCAL este format din următoarele caractere ale codului *ASCII* (*American Standard Code for Information Interchange*):

- cifrele zecimale;
- literele mari și mici ale alfabetului englez;
- semnele de punctuație;
- operatorii aritmetici și logici;
- caractere de control și editare (spațiu, sfârșit de linie sau retur de car etc.).

În unele construcții ale limbajului pot fi folosite și literele alfabetelor naționale, de exemplu literele ă, â, î, ș, ț ale alfabetului român.

1.5. VOCABULARUL LIMBAJULUI

Cele mai simple elemente, alcătuite din caractere și care au semnificație lingvistică, se numesc lexeme sau unități lexicale. Acestea formează **vocabularul** limbajului PASCAL.

Distingem următoarele **unități lexicale**:

- simboluri speciale și cuvinte-cheie;
- identificatori;
- numere;
- șiruri de caractere;
- etichete;
- directive.

1.6. SIMBOLURILE SPECIALE ȘI CUVINTELE-CHEIE

Simbolurile speciale sînt formate din unul sau două caractere:

+	plus	<	mai mic
–	minus	>	mai mare
*	asterisc	[paranteză pătrată din stînga
/	bară]	paranteză pătrată din dreapta
=	egal	(paranteză rotundă din stînga
,	virgulă)	paranteză rotundă din dreapta
:	două puncte	;	punct și virgulă
.	punct	^	putere
@	la	\$	dolar

{	acoladă din stînga	<=	mai mic sau egal
}	acoladă din dreapta	=>	mai mare sau egal
#	număr	:=	atribuire
..	puncte de suspensie	<>	neegal
(*	echivalentul acoladei {	(.	echivalentul parantezei [
*)	echivalentul acoladei }	.)	echivalentul parantezei]

Menționăm că dacă un simbol special este alcătuit din două caractere, de exemplu <= sau :=, între ele nu trebuie să apară nici un spațiu intermediar.

Cuvintele-cheie sînt formate din două sau mai multe litere:

and	și	nil	zero
array	tablou	not	nu
begin	început	of	din
case	caz	or	sau
const	constante	packed	împachetat
div	cîțul împărțirii	procedure	procedură
do	execută	program	program
downto	în descreștere la	record	articol (înregistrare)
else	altfel	repeat	repetare
end	sfîrșit	set	mulțime
file	fișier	then	atunci
for	pentru	to	la
function	funcție	type	tip
goto	treci la	until	pîna ce
if	dacă	var	variabile
in	în	while	cît
label	etichetă	with	cu
mod	restul împărțirii		

Cuvintele-cheie sînt rezervate și nu pot fi folosite în alt scop decît cel dat prin definiția limbajului.

Unitățile lexicale în studiu se definesc cu ajutorul următoarelor formule BNF:

<Simbol special> ::= + | - | * | / | = | < | > |] | [| , | (|) |
: | ; | ^ | . | @ | { | } | \$ | # | <= | => | <> | := |
.. | *<Cuvînt-cheie>* | *<Simbol echivalent>*

<Simbol echivalent> ::= (* | *) | (. | .)

<Cuvînt-cheie> ::= **and** | **array** | **begin** | **case** | **const** | **div** |
do | **downto** | **else** | **end** | **file** | **for** | **function** | **goto** | **if** |
in | **label** | **mod** | **nil** | **not** | **of** | **or** | **packed** | **procedure** |
program | **record** | **repeat** | **set** | **then** | **to** | **type** | **until** |
var | **while** | **with**

De reținut că simbolurile { , } , [și] utilizate în notația BNF sînt în același timp și elemente ale vocabularului PASCAL. Pentru a evita confuziile, aceste simboluri, ca elemente ale vocabularului, pot fi redate prin simbolurile echivalente (* , *) , (. și respectiv .) .

Întrebări și exerciții

- ❶ Memorați cuvintele-cheie ale limbajului PASCAL.
- ❷ Care este diferența dintre caractere și simboluri?
- ❸ Desenați diagramele sintactice pentru unitățile lexicale <Simbol special>, <Simbol echivalent> și <Cuvînt-cheie>.

1.7. IDENTIFICATORI

Identificatorii sînt unități lexicale care desemnează variabile, constante, funcții, programe ș. a.

Un identificator începe cu o literă, care poate fi urmată de orice combinație de litere și cifre. Lungimea identificatorilor nu este limitată, dar sînt semnificative doar primele 63 de caractere.

Amintim formulele BNF care definesc unitatea lexicală <Identificator>:

<Cifră> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Literă> ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
n | o | p | q | r | s | t | u | v | w | x | y | z

<Identificator> ::= <Literă> { <Literă> | <Cifră> }

Exemple de identificatori:

x	y	z	x1	y10	z01b	lista
listaelevilor			listatelefoanelor			
registru			adresa			
adresadomiciliu			casa10			

În construcțiile gramaticale ale limbajului PASCAL, cu excepția șirurilor de caractere, literele mari și mici se consideră echivalente. Prin urmare, sînt echivalenți și identificatorii

x și X, y și Y, z și Z, x1 și X1, y10 și Y10, z01b, Z01b, Z01B și z01B, lista, Lista, LIsta, ListA, LISTa etc.

Utilizarea literelor mari și mici ne permite să scriem identificatorii mai citeț, de exemplu:

ListaElevilor ListaTelefoanelor AdresaDomiciliu

Menționăm că în construcțiile de bază ale limbajului PASCAL nu se utilizează literele ă, â, î, ș, ț ale alfabetului român. Prin urmare, în scrierea identificatorilor semnele diacritice respective vor fi omise.

Exemple:

Suprafata	Numar	NumarElevi
Patrat	SirDeCaractere	NumarIncerari

Întrebări și exerciții

- 1 Desenați diagramele sintactice pentru unitățile gramaticale <Cifră>, <Literă> și <Identificator>.
- 2 Care din secvențele ce urmează sînt conforme definiției unității lexicale <Identificator>:

x1	Suprafata	Alx	Dreptunghi
X1	SUPRAFATA	ListaA	iI
1x	radacina	Lista1	I1j
1X	radacina	B-1	Luni
xy	R1	abc	Luna

Pentru secvențele corecte indicați drumurile respective din diagrama sintactică <Identificator>.

- 3 Găsiți perechile de identificatori echivalenți:

x101	CERCURI
ya15	SirDeCaractere
radacinaX1	Triunghiuri
radacinaX2	RegistruClasa10
triunghi	zile
cerc	X101
sirdecaractere	RegistruClasa10
registruclasa10	radaciniX1X2
COTIDIAN	RADACINAX1
ZILE	yA101

- 4 Care este destinația identificatorilor din programele PASCAL?
- 5 Pentru a găsi soluțiile x_1, x_2 ale ecuației pătrate $ax^2 + bx + c = 0$, mai întâi se calculează discriminantul d . Propuneți cîteva variante de reprezentare a coeficienților a, b, c , a discriminantului d și soluțiilor x_1, x_2 prin identificatori.

1.8. NUMERE

Numerele pot fi întregi sau reale. În mod obișnuit, se folosește sistemul zecimal de numerație. În fig. 1.4 sînt prezentate diagramele sintactice pentru unitățile lexicale <Număr întreg> și <Număr real>.

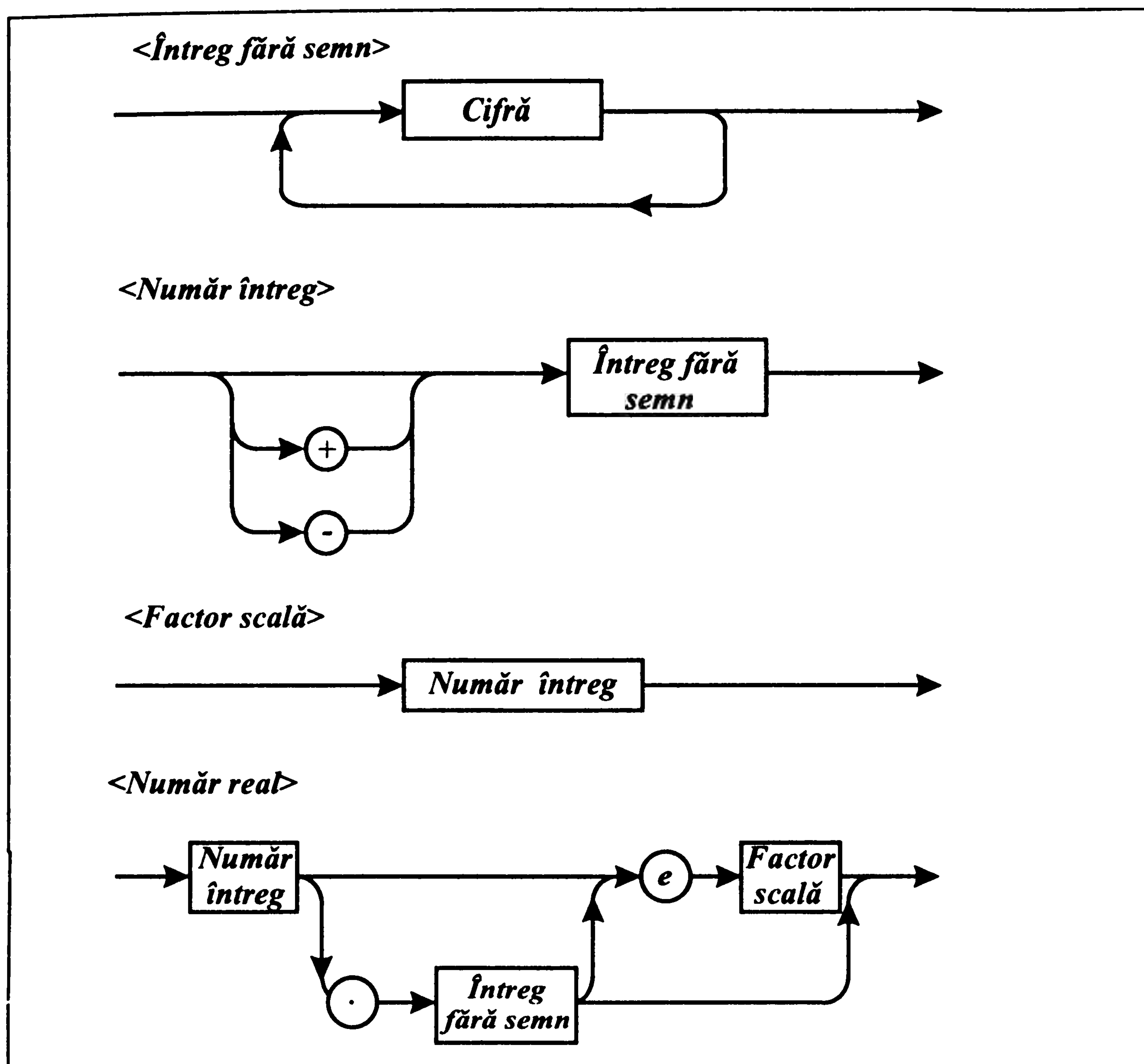


Fig. 1.4. Diagramele sintactice <Număr întreg> și <Număr real>

Exemple de numere întregi:

23	-23	0023	+023	-0023
318	00318	+0318	-318	-0318
1996	+1996	-1996	0001996	+001996

În cazul numerelor reale partea fracționară se separă de partea întreagă prin punct. Punctul zecimal trebuie să fie precedat și urmat de cel puțin o cifră zecimală.

Exemple de numere reale:

3.1415	+3.04	0.0001
283.19	-3.04	-0.0001
-256.19	28.17	+0.0001

În scrierea numerelor reale se poate utiliza și un **factor de scală**. Acesta este un număr întreg precedat de litera e (sau E) și indică că

numărul urmat de factorul de scală se înmulțește cu 10 la puterea respectivă.

Exemple:

<u>forma uzuală</u>	<u>notația PASCAL</u>
$8,12 \cdot 10^{-5}$	$8.12e-5$
$749,512 \cdot 10^8$	$749.512e+8$
$-0,0823 \cdot 10^{-12}$	$-0.0823e-12$
$3250,4 \cdot 10^6$	$3250.4e06$
$3,421 \cdot 10^{16}$	$3.421e16$

Evident, $8.12e-05$, $812e-7$, $0.812e-4$, $81.2e-6$ reprezintă una și aceeași valoare $8,12 \cdot 10^{-5}$.

Întrebări și exerciții

- ❶ Care din secvențele de caractere ce urmează sînt conforme definiției unității lexicale *<Număr întreg>*?

-418	2469	32,14	-621	+621
0-418	-6210	+00621	32,014	-00418
621+	0+2469	24690	-719	-00621

Găsiți numerele întregi care reprezintă una și aceeași valoare.

- ❷ Pornind de la diagramele sintactice din *fig. 1.4*, scrieți formulele BNF pentru definirea unității lexicale *<Număr întreg>*.
- ❸ Care din secvențele ce urmează sînt conforme definiției unității lexicale *<Număr real>*?

3.14	281.3	0,618284e00
2.514e+5	591328	1961.
591328E+3	2514e+2	28130E-2
.000382	-464.597e+3	591.328
0.1961E+4	+519.328e-4	-658.14e-6
+314629.	591328e-3	2514e+2
0.000314E4	28130e-2	618.248e-3

Găsiți numerele reale care reprezintă una și aceeași valoare. Scrieți aceste numere în forma uzuală.

- ❹ Pornind de la diagramele sintactice din *fig. 1.4*, scrieți formulele BNF pentru definirea unității lexicale *<Număr real>*.
- ❺ Indicați pe diagramele sintactice din *fig. 1.4* drumurile care corespund următoarelor numere:

-418	281.3	32.014	+0001
1961.0	2.514e+5	591.328	-614.85e-3
2514E+2	-1951.12	+19.511e+2	2013e-4

1.9. ȘIRURI DE CARACTERE

Șirurile de caractere sînt șiruri de caractere imprimabile, delimitate de apostrof. În șirul de caractere apostroful apare dublat. Accentuăm că în cazul șirurilor de caractere literele mari și mici apar drept caractere distincte.

Exemple:

'Variabila x'
'Calculul aproximativ'
'APOSTROFUL ' ' ESTE DUBLAT'

Spre deosebire de alte unități lexicale ale limbajului PASCAL, în șirurile de caractere pot fi utilizate și literele ă, â, î, ș, ț ale alfabetului român. Pentru aceasta e necesar ca pe calculatorul la care lucrați să fie instalate **programele-pilot** ce asigură introducerea, afișarea și imprimarea literelor în studiu.

Exemple:

'Șir de caractere'
'Limba engleză'
'Suprafață'
'Număr încercări'

Unitatea lexicală <Șir de caractere> se definește cu ajutorul următoarelor formule BNF:

<Șir de caractere> ::= ' <Element șir> {<Element șir>} '

<Element șir> ::= ' ' | <Orice caracter imprimabil>

Diagrama sintactică a unității lexicale în studiu este prezentată în fig. 1.5.

Întrebări și exerciții

- ❶ Indicați pe diagramele sintactice din fig. 1.5 drumurile care corespund următoarelor șiruri de caractere:

'variabila z'
'''
'Caracterele ''x'', ''y''
'UNITĂȚI LEXICALE'

- ❷ Care dintre secvențele ce urmează sînt conforme definiției unității lexicale <Șir de caractere>:

'Număr întreg'	'Anul 1997'
'Sfîrșitul programului'	'Rădăcină pătrată'
'APOSTROF'	'Anul '97'

''x''
'funcție'

'Lista telefoanelor'
''

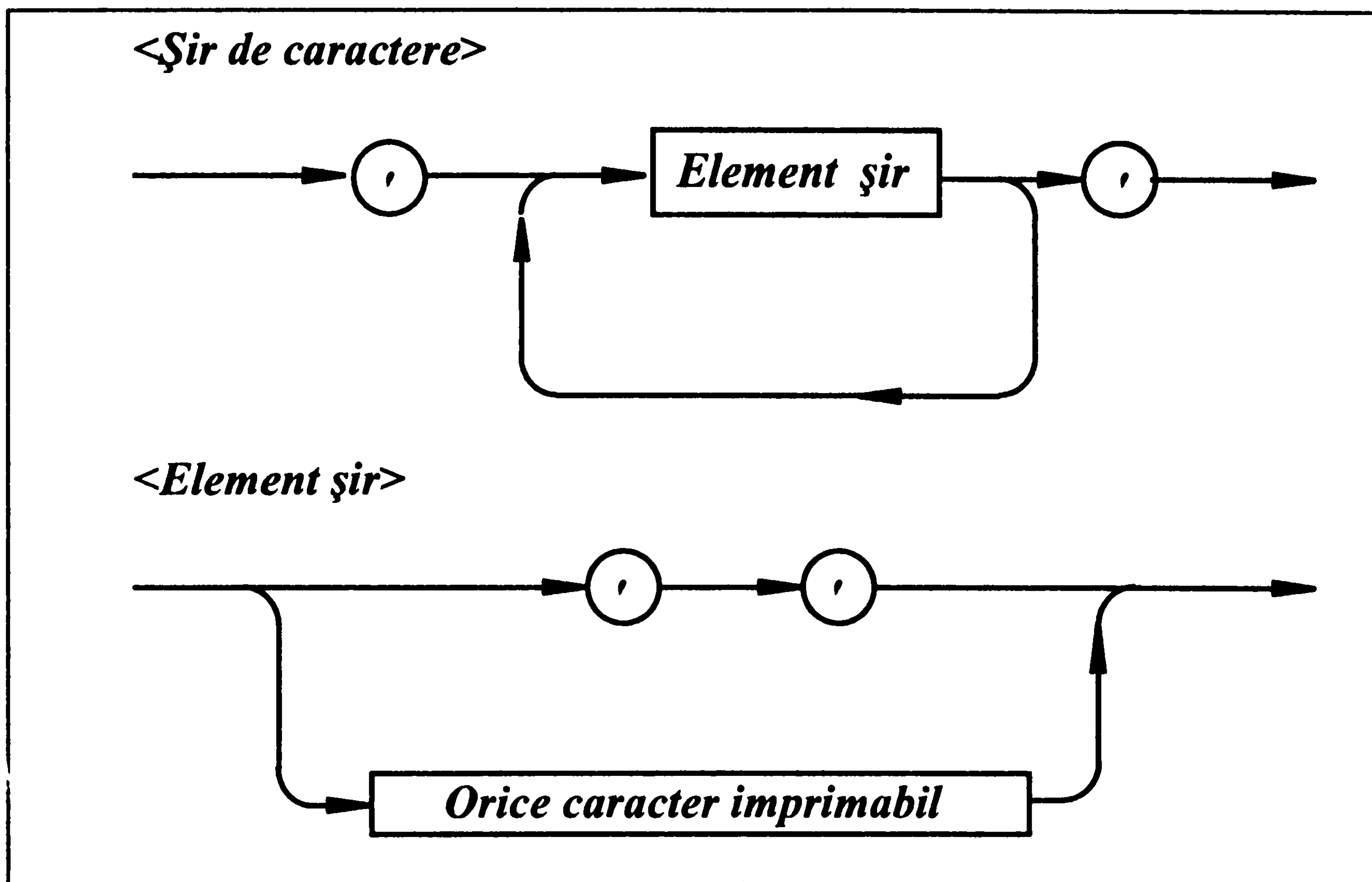


Fig. 1.5. Diagrama sintactică <Șir de caractere>

1.10. ETICHETE

Etichetele sînt numere întregi fără semn din domeniul 0, 1, ..., 9999 și se utilizează pentru a marca instrucțiunile limbajului PASCAL.

Exemple:

1 100 999 582 1004

Evident, formula BNF care definește unitatea lexicală în studiu are forma:

$\langle \text{Etichetă} \rangle ::= \langle \text{Întreg fără semn} \rangle$

1.11. DIRECTIVE

Unitatea lexicală <Directivă> se definește exact ca identificatorul:

$\langle \text{Directivă} \rangle ::= \langle \text{Literă} \rangle \{ \langle \text{Literă} \rangle \mid \langle \text{Cifră} \rangle \}$

Efectiv, directivele sînt cuvinte rezervate care au o semnificație specială. Limbajul-standard conține o singură directivă:

forward

Aceasta se utilizează la descrierea unor proceduri (subalgoritmi) și funcții definite de utilizator.

1.12. SEPARATORI

Orice program PASCAL constă din lexeme și separatori. Separatorii folosiți în limbaj sînt spațiul, sfîrșitul de linie (retur de car) și comentariul.

La scrierea consecutivă a identificatorilor, a cuvintelor-cheie, a numerelor fără semn și a directivelor, începutul unei unități lexicale ar putea fi interpretat în unele cazuri drept o continuare a celei precedente. Pentru a delimita unitățile lexicale în studiu, între ele se inserează spații sau returnuri de car.

Exemple:

```
x div y
not x
begin
writeln(x);
writeln(y);
end.
```

Menționăm că simbolurile speciale compuse din două caractere <=, =>, <>, :=, .. etc., identificatorii, numerele ș.a.m.d. sînt unități lexicale ale programului. Prin urmare, nu se pot introduce spații sau returnuri de car între caracterele componente.

Exemple:

Corect

```
CitireDisc
Program
:=
..
345
downto
begin
```

Inc corect

```
Citire Disc
Pro gram
: =
. .
3 45
down to
be gin
```

Comentariile sînt secvențe de caractere precedate de { și urmate de } .

Exemple:

```
{ Program elaborat de Radu Ion }  
{ Introducerea datelor inițiale }  
{ Datele inițiale se introduc de la tastatură.  
Rezultatele vor fi afișate pe ecran și tipărite la  
imprimantă peste 3-4 minute }
```

Comentariile nu influențează în nici un fel derularea programelor PASCAL și se utilizează pentru a include în ele unele precizări, explicații, informații suplimentare etc. Evident, comentariile sînt predestinate nu calculatorului, ci persoanelor care citesc programul respectiv.

Accentuăm că utilizarea rațională a comentariilor, spațiilor și retururilor de car asigură scrierea unor programe lizibile (ușor de citit).

Capitolul 2

.....

TIPURI DE DATE SIMPLE

2.1. CONCEPTUL DE DATĂ

Informația ce urmează să fie supusă unei prelucrări este accesibilă calculatorului în formă de date. **Datele** sînt constituite din cifre, litere, semne, numere, șiruri de caractere ș.a.m.d.

Într-un limbaj cod-calculator datele sînt reprezentate prin secvențe de cifre binare. De exemplu, la nivelul procesorului numărul natural 1039 se reprezintă în sistemul de numerație binar ca:

10000001111

Într-un program PASCAL datele sînt reprezentate prin **mărimi**, și anume, prin variabile și constante. Pentru a scuti utilizatorul de toate detaliile legate de reprezentarea internă a datelor, în PASCAL se utilizează diverse tipuri de date.

Prin **tip de date** se înțelege o **mulțime de valori** și o **mulțime de operații** care pot fi efectuate cu valorile respective.

De exemplu, în versiunea *Turbo PASCAL 7.0* tipul `integer` include mulțimea numerelor întregi

$\{-32768, -32767, \dots, -2, -1, 0, 1, 2, \dots, 32767\}$.

Cu aceste numere pot fi efectuate următoarele operații:

+ adunarea;
- scăderea;
* înmulțirea;
mod restul împărțirii;
div cîțul împărțirii ș.a.

Tipul de date `real` (`real`) include o submulțime a numerelor reale, operațiile +, -, *, / (împărțirea) ș.a.

Operațiile `mod` și `div`, admise în tipul de date `integer`, sînt inadmisibile în tipul de date `real`.

Conceptul de dată realizat în limbajul PASCAL presupune:

1) fiecare mărime (variabilă sau constantă) într-un program în mod obligatoriu se asociază cu un anumit tip de date;

2) tipul unei variabile definește mulțimea de valori pe care le poate lua variabila și operațiile care pot fi efectuate cu aceste valori;

3) există tipuri de date de interes general, definiția cărora se consideră cunoscută: `integer`, `real`, `char` (caracter), `boolean` (logic), `text` ș.a.;

4) pe baza tipurilor cunoscute programatorul poate crea tipuri noi, adecvate informațiilor de prelucrat.

Tipul variabilelor se declară explicit cu ajutorul cuvântului-cheie **var**.

Exemplu:

```
Var x, y : integer;  
      z : real;
```

Tipul unei **constante** se declară implicit prin forma ei textuală.

De exemplu, 10 este un număr de tip `integer`, iar 10.0 este un număr de tip `real`.

Întrebări și exerciții

- ❶ Cum se reprezintă datele în limbajul cod-calculator? Care sînt avantajele și deficiențele acestei reprezentări?
- ❷ Cum se reprezintă datele într-un program PASCAL? Care este diferența dintre variabile și constante?
- ❸ Explicați semnificația termenului *tip de date*. Dați exemple.
- ❹ Cum se asociază o variabilă la un anumit tip de date?
- ❺ Determinați tipul variabilelor `r`, `s`, `t`, `x`, `y` și `z` din declarația ce urmează:

```
var  r, y : integer;  
      s, z : real;  
      t, x : boolean;
```

- ❻ Scrieți o declarație care ar defini `a`, `b` și `c` ca variabile întregi, iar `p` și `q` ca variabile *text*.

- ❼ Precizați tipul următoarelor numere:

-301; -301.0; +6100; -61.00e+2; 3.14.

2.2. TIPUL DE DATE `integer`

Mulțimea de valori ale tipului de date `integer` este formată din numerele întregi care pot fi reprezentate pe calculatorul-gazdă al limbajului. Valoarea maximă poate fi referită prin constanta `MaxInt`, cunoscută oricărui program PASCAL. De obicei, valoarea minimă, admisă de tipul de date în studiu, este `-MaxInt` sau `-(MaxInt + 1)`.

Programul ce urmează afișează pe ecran valoarea constantei predefinite MaxInt.

```
Program P2;  
{ Afișarea constantei predefinite MaxInt }  
begin  
  writeln('MaxInt=', MaxInt);  
end.
```

Pe un calculator IBM PC, versiunea Turbo PASCAL 7.0, constanta MaxInt are valoarea 32767, iar mulțimea de valori ale tipului integer este:

$\{-32768, -32767, \dots, -2, -1, 0, 1, 2, \dots, 32767\}$.

Operațiile care se pot face cu valorile întregi sînt: +, -, *, mod, div ș.a. Rezultatele acestor operații pot fi vizualizate cu ajutorul programului P3.

```
Program P3;  
{ Operații cu date de tipul integer }  
var x, y, z : integer;  
begin  
  writeln('Introduceți numerele întregi x, y:');  
  readln(x,y);  
  writeln('x=', x);  
  writeln('y=', y);  
  z:=x+y; writeln('x+y=', z);  
  z:=x-y; writeln('x-y=', z);  
  z:=x*y; writeln('x*y=', z);  
  z:=x mod y; writeln('x mod y=', z);  
  z:=x div y; writeln('x div y=', z);  
end.
```

Evident, rezultatele operațiilor +, -, * cu valori întregi trebuie să aparțină mulțimii de valori ale tipului de date integer. Dacă programatorul nu acordă atenția cuvenită acestei reguli, apar erori de depășire. Aceste erori vor fi semnalate în procesul compilării sau execuției programului respectiv. Pentru exemplificare prezentăm programele P4 și P5:

```
Program P4;  
{ Eroare de depășire semnalată în procesul  
  compilării }  
var x : integer;  
begin  
  x:=MaxInt+1; { Eroare, x>MaxInt }  
  writeln(x);  
end.
```

```
Program P5;  
{ Eroare de depășire semnalată în procesul execuției}  
var x, y : integer;  
begin  
  x:=MaxInt;  
  y:=x+1; { Eroare, y>MaxInt }  
  writeln(y);  
end.
```

Întrebări și exerciții

- ❶ Care este mulțimea de valori ale tipului de date integer? Ce operații se pot face cu aceste valori?
- ❷ Când apar erori de depășire? Cum se depistează aceste erori?
- ❸ Determinați valoarea constantei MaxInt a versiunii PASCAL cu care lucrați D-voastră.
- ❹ Se consideră programele :

```
Program P6;  
{ Eroare de depășire }  
var x : integer;  
begin  
  x:=-2*MaxInt;  
  writeln(x);  
end.
```

```
Program P7;  
{ Eroare de depășire }  
var x, y : integer;  
begin  
  x:=-MaxInt;  
  y:=x-10;  
  writeln(y);  
end.
```

Când vor fi semnalate erori de depășire: la compilare sau la execuție?

- ❺ Dați exemple de valori ale variabilelor x și y din programul P3 pentru care apar erori de depășire.

2.3. TIPUL DE DATE real

Mulțimea de valori ale tipului de date în studiu este formată din numerele reale care pot fi reprezentate pe calculatorul-gazdă al limbajului.

De exemplu, în versiunea Turbo PASCAL 7.0 domeniul de valori al tipului real este $2,9 \cdot 10^{-39}$, ..., $1,7 \cdot 10^{38}$, numerele fiind reprezentate cu o precizie de 11–12 cifre zecimale.

În programul ce urmează variabilelor reale x , y și z li se atribuie valorile, respectiv, 1,1, $-6,14 \cdot 10^8$ și $90,3 \cdot 10^{-29}$, ulterior afișate pe ecran.

```
Program P8;  
{ Date de tip real }  
var x, y, z : real;  
begin  
  x:=1.1;  
  y:=-6.14e8;  
  z:=90.3e-29;  
  writeln('x=', x);  
  writeln('y=', y);  
  writeln('z=', z);  
end.
```

Amintim că la scrierea numerelor reale virgula zecimală este redată prin punct, iar puterea lui 10 – prin factorul de scală (vezi paragraful 1.8).

Operațiile care se pot face cu valorile reale sînt +, -, *, / (împărțirea) ș.a.

Operațiile asupra valorilor reale sînt în general aproximative datorită erorilor de rotunjire. Evident, rezultatele operațiilor în studiu trebuie să aparțină domeniului de valori ale tipului de date `real`. În caz contrar, apar erori de depășire.

Proprietățile operațiilor +, -, * și / pot fi studiate cu ajutorul programului ce urmează:

```
Program P9;  
{ Operații cu date de tipul real }  
var x, y, z : real;  
begin  
  writeln('Introduceți numerele reale x, y:');  
  readln(x,y);  
  writeln('x=', x);  
  writeln('y=', y);  
  z:=x+y; writeln('x+y=', z);  
  z:=x-y; writeln('x-y=', z);  
  z:=x*y; writeln('x*y=', z);  
  z:=x/y; writeln('x/y=', z);  
end.
```

În *tabelul 2.1* sînt prezentate datele afișate de programul P9 (versiunea Turbo PASCAL 7.0) pentru unele valori ale variabilelor x și y . Se observă că rezultatele operațiilor $x+y$ și $x-y$ din primele două linii ale *tabelului 2.1* sînt exacte. În cazul valorilor $x = 1,0$, $y = 1,0 \cdot 10^{-11}$ (linia 3 a tabelului în studiu) rezultatul adunării este aproxi-

mativ, iar cel al scăderii – exact. Ambele rezultate din linia a patra sînt aproximative. În cazul valorilor $x = y = 1,7 \cdot 10^{38}$ (linia 5) are loc o depășire la efectuarea adunării. Pentru valorile $x = 3,1 \cdot 10^{-39}$, $y = 3,0 \cdot 10^{-39}$ (linia 6) rezultatul adunării este exact, iar rezultatul scăderii este aproximativ.

Tabelul 2.1. Rezultatele programului P9

Nr. crt.	x	y	x+y	x-y
1	1,0	1,0	2.0000000000 E+00	0.0000000000 E+00
2	1,0	$1 \cdot 10^{10}$	1.0000000001 E+00	9.9999999990 E-01
3	1,0	$1,0 \cdot 10^{11}$	1.0000000000 E+00	9.9999999999 E-01
4	1,0	$1,0 \cdot 10^{12}$	1.0000000000 E+00	1.0000000000 E+00
5	$1,7 \cdot 10^{38}$	$1,7 \cdot 10^{38}$	depășire	0.0000000000 E+00
6	$3,1 \cdot 10^{-39}$	$3,0 \cdot 10^{-39}$	6.1000000000 E-39	0.0000000000 E+00

Însumîndu-se, erorile de calcul, proprii tipului de date real, pot compromite rezultatele execuției unui program. Evaluarea și, dacă e necesar, suprimarea erorilor semnificative cade în sarcina programatorului.

Întrebări și exerciții

- 1 Cum se scriu numerele reale în limbajul PASCAL?
- 2 Determinați domeniul de valori ale tipului de date real din versiunea PASCAL cu care lucrați. Care este precizia numerelor respective?
- 3 Ce operații se pot face cu datele de tip real? Sînt oare exacte aceste operații?
- 4 Lansați în execuție programul P9 pentru următoarele valori ale variabilelor x, y:
 - a) $x = 2,0;$ $y = -3,0;$
 - b) $x = 143 \cdot 10^2;$ $y = 153 \cdot 10^{-3};$
 - c) $x = 3,0;$ $y = 2,0 \cdot 10^{12};$
 - d) $x = 3,0;$ $y = 2,0 \cdot 10^{-12};$
 - e) $x = 2,9 \cdot 10^{-39};$ $y = 6,4 \cdot 10^{-3};$
 - f) $x = 7,51 \cdot 10^{21};$ $y = -8,64 \cdot 10^{17};$
 - g) $x = 1,0;$ $y = 2,9 \cdot 10^{-39}.$

Verificați rezultatele operațiilor respective. Explicați mesajele afișate pe ecran.

- 5 Care sînt cauzele erorilor de calcul cu date de tip real?

2.4. TIPUL DE DATE boolean

Tipul de date boolean (logic) include valorile de adevăr false (fals) și true (adevărat). În programul ce urmează variabilei `x` i se atribuie consecutiv valorile false și true, afișate ulterior pe ecran.

```
Program P10;
{ Date de tip boolean }
var x : boolean;
begin
  x:=false;
  writeln(x);
  x:=true;
  writeln(x);
end.
```

Operațiile predefinite ale tipului de date boolean sînt:

- not** negația (inversia logică, operația logică *NU*);
- and** conjuncția (produsul logic, operația logică *ȘI*);
- or** disjuncția (suma logică, operația logică *SAU*).

Tabelele de adevăr ale operațiilor în studiu sînt prezentate în *fig. 2.1*.

<i>x</i>	not <i>x</i>
<i>false</i>	<i>true</i>
<i>true</i>	<i>false</i>

<i>x</i>	<i>y</i>	<i>x</i> and <i>y</i>
<i>false</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>false</i>
<i>true</i>	<i>false</i>	<i>false</i>
<i>true</i>	<i>true</i>	<i>true</i>

<i>x</i>	<i>y</i>	<i>x</i> or <i>y</i>
<i>false</i>	<i>false</i>	<i>false</i>
<i>false</i>	<i>true</i>	<i>true</i>
<i>true</i>	<i>false</i>	<i>true</i>
<i>true</i>	<i>true</i>	<i>true</i>

Fig. 2.1. Tabelele de adevăr ale operațiilor logice **not**, **and** și **or**

Proprietățile operațiilor logice not, and și or pot fi studiate cu ajutorul programului P11.

```
Program P11;
{Operații cu date de tip boolean }
var x, y, z : boolean;
begin
  x:=false; y:=false;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
  x:=false; y:=true;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
  x:=true; y:=false;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
  x:=true; y:=true;
  writeln('x=', x, 'y=', y);
  z:=not x; writeln('not x = ', z);
  z:=x and y; writeln('x and y = ', z);
  z:=x or y; writeln('x or y = ', z);
  writeln;
end.
```

Spre deosebire de variabilele de tip întreg sau real, valorile curente ale variabilelor booleene nu pot fi citite de la tastatură cu ajutorul procedurii-standard readln. Din acest motiv, în programul P11 valorile curente ale variabilelor x și y sînt date prin atribuire.

Întrebări și exerciții

- ❶ Numiți mulțimea de valori și operațiile cu date de tip boolean.
- ❷ Memorizați tabelele de adevăr ale operațiilor logice.
- ❸ Elaborați un program care afișează pe ecran tabelul de adevăr al operației logice not.
- ❹ Elaborați un program care calculează valorile funcției logice $z = x \ \& \ y$ pentru toate valorile posibile ale argumentelor x, y.
- ❺ Elaborați un program care afișează valorile funcției logice $z = x \ \vee \ y$.

2.5. TIPUL DE DATE **char**

Mulțimea valorilor acestui tip de date este o mulțime finită și ordonată de caractere. Valorile în studiu se desemnează prin includerea fiecărui caracter între două semne ' (apostrof), de exemplu, 'A', 'B', 'C' etc. Însuși apostroful se dublează, reprezentându-se prin ''.

În programul ce urmează variabilei *x* de tip **char** i se atribuie consecutiv valorile 'A', '+' și '', afișate ulterior pe ecran.

```
Program P12;  
{ Date de tip char }  
var x : char;  
begin  
    x:='A';  
    writeln(x);  
    x:='+';  
    writeln(x);  
    x:='';  
    writeln(x);  
end.
```

Valorile curente ale unei variabile de tip **char** pot fi citite de la tastatură cu ajutorul procedurii-standard **readln**. Pentru exemplificare prezentăm programul P13 care citește de la tastatură și afișează pe ecran valori de tipul **char**.

```
Program P13;  
{ Citirea și afișarea caracterelor }  
var x : char;  
begin  
    readln(x); writeln(x);  
    readln(x); writeln(x);  
    readln(x); writeln(x);  
end.
```

Caracterele respective se introduc de la tastatură și se afișează pe ecran fără apostrofurile care le încadrează în textul unui program PASCAL.

De regulă, caracterele unei versiuni concrete a limbajului PASCAL sînt **ordonate** conform tabelului de cod *ASCII* (vezi paragraful 1.4).

Numărul de ordine al oricărui caracter din mulțimea de valori ale tipului **char** poate fi aflat cu ajutorul funcției predefinite **ord**. De exemplu:

```
ord('A')=65,  
ord('B')=66,
```

`ord('C')=67`

ș.a.m.d.

Programul ce urmează afișează pe ecran numărul de ordine al patru caractere citite de la tastatură.

```
Program P14;  
{ Studierea funcției ord }  
var x : char; { caracter }  
    i : integer; { număr de ordine }  
begin  
    readln(x); i:=ord(x); writeln(i);  
    readln(x); i:=ord(x); writeln(i);  
    readln(x); i:=ord(x); writeln(i);  
    readln(x); i:=ord(x); writeln(i);  
end.
```

Funcția predefinită `chr` returnează caracterul care corespunde numărului de ordine indicat. Evident,

`chr(65)='A',`

`chr(66)='B',`

`chr(67)='C'`

ș.a.m.d.

Programul P15 afișează pe ecran caracterele ce corespund numerelor de ordine citite de la tastatură.

```
Program P15;  
{ Studierea funcției chr }  
var i : integer; { număr de ordine }  
    x : char; { caracter }  
begin  
    readln(i); x:=chr(i); writeln(x);  
    readln(i); x:=chr(i); writeln(x);  
    readln(i); x:=chr(i); writeln(x);  
    readln(i); x:=chr(i); writeln(x);  
end.
```

Amintim că un set extins *ASCII* include 256 de caractere, numerotate cu 0, 1, 2, ..., 255.

Tipul de date `char` se utilizează pentru formarea unor structuri de date mai complexe, în particular, a șirurilor de caractere.

Întrebări și exerciții

- ❶ Care este mulțimea de valori ale tipului de date `char`?
- ❷ Cum este ordonată mulțimea de valori ale tipului `char`?
- ❸ Determinați numerele de ordine ale următoarelor caractere:

- cifrele zecimale;
 - literele mari ale alfabetului englez;
 - semnele de punctuație;
 - operatorii aritmetici și logici;
 - caracterele de control și editare;
 - literele alfabetului român (dacă sînt implementate pe calculatorul D-voastră).
- ④ Determinați caracterele care corespund următoarelor numere de ordine:
77, 109, 79, 111, 42, 56, 91, 123.
- ⑤ Elaborați un program care afișează pe ecran setul de caractere al calculatorului cu care lucrați.

2.6. TIPURI DE DATE *ENUMERARE*

Tipurile *integer*, *real*, *boolean* și *char*, studiate pînă acum, sînt tipuri predefinite, cunoscute oricărui program PASCAL. În completare la tipurile predefinite, programatorul poate defini și utiliza tipuri proprii de date, în particular, tipuri *enumerare*.

Un tip *enumerare* include o mulțime ordonată de valori specificate prin identificatori. Denumirea unui tip de date *enumerare* și mulțimea lui de valori se indică în partea declarativă a programului după cuvîntul-cheie **type** (tip).

Exemplu:

```
type
  Culoare      = (Galben, Verde, Albastru, Violet);
  Studii       = (Elementare, Medii, Superioare);
  Raspuns      = (Nu, Da);
```

Primul identificator din lista de enumerare desemnează cea mai mică valoare, cu numărul de ordine zero. Identificatorul al doilea va avea numărul de ordine unu, al treilea – numărul doi ș.a.m.d. Numărul de ordine al unei valori poate fi aflat cu ajutorul funcției predefinite *ord*.

Exemple:

```
ord(Galben)=0,
ord(Verde)=1,
ord(Albastru)=2,
ord(Violet)=3,
ord(Elementare)=0,
ord(Medii)=1
ș.a.m.d.
```


Programul ce urmează afișează pe ecran numerele de ordine ale valorilor tipului de date *Studii*.

```
Program P16;
{ Tipul de date Studii }
type Studii=(Elementare, Medii, Superioare);
var i : integer; { număr de ordine }
begin
    i:=ord(Elementare); writeln(i);
    i:=ord(Medii); writeln(i);
    i:=ord(Superioare);
    writeln(i);
end.
```

Variabilele de tip *enumerare* se declară cu ajutorul cuvîntului-cheie **var**. Ele pot lua numai valori din lista de enumerare a tipului de date cu care sînt asociate.

În programul ce urmează variabila *x* ia valoarea Albastru; variabila *y* ia valoarea Nu. Numerele de ordine ale acestor valori se afișează pe ecran.

```
Program P17;
{ Variabile de tip enumerare }
type Culoare=(Galben, Verde, Albastru, Violet);
    Raspuns=(Nu, Da);
var x : Culoare; { variabilă de tip Culoare }
    y : Raspuns; { variabilă de tip Raspuns }
    i : integer; { număr de ordine }
begin
    x:=Albastru;
    i:=ord(x); writeln(i);
    y:=Nu; i:=ord(y); writeln(i);
end.
```

În cazurile în care într-un program PASCAL se definesc mai multe tipuri de date, listele de enumerare nu trebuie să conțină identificatori comuni.

De exemplu, declarația

```
type Studii=(Elementare, Medii, Superioare);
    Grade=(Inferioare, Superioare)
```

este incorectă, întrucît identificatorul *Superioare* apare în ambele liste.

Valorile curente ale variabilelor de tip *enumerare* nu pot fi citite de la tastatură sau afișate pe ecran cu ajutorul procedurilor-standard `readln` și `writeln`. Totuși utilizarea tipurilor de date în studiu permite elaborarea unor programe lizibile, simple și eficiente.

Întrebări și exerciții

- ❶ Cum se definește un tip de date *enumerare*? Care este mulțimea de valori ale unui tip *enumerare*?
- ❷ Contează oare ordinea în care apar identificatorii într-o listă de enumerare?
- ❸ Elaborați un program care afișează pe ecran numerele de ordine ale valorilor următoarelor tipuri de date:

```
Continente= (Europa,Asia,Africa,AmericaDeNord,  
             AmericaDeSud,Australia,Antarctida);  
Sex=(Barbat,Femeie);  
PuncteCardinale=(Nord,Sud,Est,Vest);  
Etaje=(Unu,Doi,Trei,Patru,Cinci).
```

- ❹ Numiți tipul fiecărei variabile din programul P18:

```
Program P18;  
type Litere=(A, B, C, D, E, F, G);  
var x : Litere; y : char; i : integer;  
begin  
    x:=A; i:=ord(x); writeln(i);  
    y:='A'; i:=ord(y); writeln(i);  
end.
```

Ce va afișa pe ecran acest program?

- ❺ Se consideră declarațiile:

```
type Culoare=(Galben, Verde, Albastru, Violet);  
      Fundal=(Alb, Negru, Gri);  
var  x, y : Culoare;  
      z : Fundal;
```

Care din instrucțiunile ce urmează sînt corecte?

- | | |
|--------------|-----------------|
| a) x:=Verde; | e) y:=Gri; |
| b) y:=Negru; | f) z:=Violet; |
| c) z:=Alb; | g) x:=Albastru; |
| d) x:=Gri; | h) y:=Azuriu; |

- ❻ Inserați înainte de cuvîntul-cheie end al programului P17 una din liniile ce urmează:

```
readln(x);  
writeln(x);
```

Explicați mesajele afișate pe ecran în procesul compilării programului modificat.

2.7. TIPURI DE DATE *SUBDOMENIU*

Un tip de date *subdomeniu* include o submulțime de valori ale unui tip deja definit, denumit tip de bază. Tipul de bază trebuie să fie *integer*, *boolean*, *char* sau *enumerare*.

Denumirea unui tip de date *subdomeniu*, valoarea cea mai mică și valoarea cea mai mare (în sensul numărului de ordine) se indică în partea declarativă a programului după cuvântul-cheie **type**.

Exemple:

```
1)  type Indice=1..10;  
      Litera='A'..'Z';  
      Cifra='0'..'9';
```

Tipul *Indice* este un *subdomeniu* al tipului predefinit *integer*. Tipurile *Litera* și *Cifra* sînt *subdomenii* ale tipului predefinit *char*.

```
2)  type Zi=(L, Ma, Mi, J, V, S, D);  
      ZiDeLucru=L..V;  
      ZiDeOdihna=S..D;
```

Tipurile *ZiDeLucru* și *ZiDeOdihna* sînt *subdomenii* ale tipului *enumerare* *Zi*, definit de utilizator.

```
3)  type T1=(A, B, C, D, E, F, G, H);  
      T2=A..F;  
      T3=C..H;
```

Tipurile *T2* și *T3* sînt *subdomenii* ale tipului *enumerare* *T1*.

Tipurile de bază ale tipurilor de date *subdomeniu* din exemplele în studiu sînt:

<u>tip enumerare</u>	<u>tipul de bază</u>
Indice	integer
Litera	char
Cifra	char
ZiDeLucru	Zi
ZiDeOdihna	Zi
T2	T1
T3	T1

Variabilele unui tip de date *subdomeniu* se declară cu ajutorul cuvîntului-cheie *var*. O variabilă de tip *subdomeniu* moștenește toate proprietățile variabilelor tipului de bază, dar valorile ei trebuie să fie numai din intervalul specificat. În caz contrar este semnalată o eroare și programul se oprește.

Exemplu:

```
Program P19;  
{ Valorile variabilelor de tip subdomeniu }  
type Indice=1..10;  
      Zi=(L, Ma, Mi, J, V, S, D);  
      ZiDeLucru=L..V;  
      ZiDeOdihna=S..D;  
var i   : Indice; { valori posibile: 1,2, ...,10 }  
      z   : Zi;     { valori posibile: L,Ma, ...,D }  
      z1  : ZiDeLucru;{ valori posibile: L,Ma, ...,V }  
      zo  : ZiDeOdihna;{ valori posibile: S, D }  
begin  
  i:=5; i:=11; { Eroare, i>10 }  
  z:=L; z1:=J; z1:=S; { Eroare, z1>V }  
  zo:=S; zo:=V; { Eroare, zo<S }  
  writeln('Sfîrşit');  
end.
```

Programul P20 demonstrează cum tipul Pozitiv moşteneşte proprietăţile tipului de bază integer.

```
Program P20;  
{ Tipul Pozitiv moşteneşte proprietăţile tipului }  
integer }  
type Pozitiv=1..32767;  
var x, y, z : Pozitiv;  
begin  
  writeln('Introduceţi numerele pozitive x, y:');  
  readln(x,y);  
  writeln('x=', x);  
  writeln('y=', y);  
  z:=x+y; writeln('x+y=', z);  
  z:=x-y; writeln('x-y=', z);  
  z:=x*y; writeln('x*y=', z);  
  z:=x mod y; writeln('x mod y=', z);  
  z:=x div y; writeln('x div y=', z);  
end.
```

Se observă că operaţiile +, -, *, mod şi div ale tipului de bază integer sînt moştenite de tipul *enumerare* Pozitiv. Dar, spre deosebire de variabilele de tip integer, variabilele de tip Pozitiv nu pot lua valori negative.

Utilizarea tipurilor de date *subdomeniu* face programele mai intuitive şi simplifică verificarea lor.

Subliniem faptul că în limbajul PASCAL nu este permisă definirea unui subdomeniu al tipului real, deoarece valorile acestuia nu au numere de ordine.

Întrebări și exerciții

❶ Cum se definește un tip *subdomeniu*? Care este mulțimea de valori ale unui tip *subdomeniu*?

❷ Numiți tipul de bază al fiecărui tip *subdomeniu*:

```
type T1=(A,B,C,D,E,F,G,H) ;  
      T2=-60..60;  
      T3=5..9;  
      T4='5'..'9';  
      T5=A..E;  
      T6='A'..'E';
```

❸ Ce valori poate lua fiecare variabilă din următoarele declarații:

```
type T1=(A,B,C,D,E,F,G,H) ;  
      T2=1..9;  
      T3=6..15;  
      T4=-100..100;  
      T5='A'..'Z';  
      T6='0'..'9';  
      T7=C..F;  
var  i:integer;  
      j:T2;  
      k:T3;  
      m:T4;  
      q:char;  
      p:T5;  
      r:T6;  
      s:T1;  
      t:T7;
```

Numiți tipul de bază al fiecărui tip *subdomeniu*. Indicați setul de operații moștenit de la tipul de bază.

❹ Care din următoarele definiții sînt corecte? Argumentați răspunsul.

a) **type** Lungime=1.0e-2..1.0;
 Latime=1.0e-2..0.5;

b) **type** Indice=1..10;
 Abatere=+5..-5;
 Deviere=-10..+10;

c) **type** T1=(A,B,C,D,E,F,G,H) ;
 T2=C..H;
 T3=F..B;

d) **type** Luni= (Ianuarie,Februarie,Martie,Aprilie,
 Mai,Iunie,Iulie,August,Septembrie,
 Octombrie,Noiembrie,Decembrie) ;
 LuniDeIarna=(Decembrie..Februarie) ;
 LuniDePrimavara=(Martie..Mai) ;
 LuniDeVara=(Iunie..August) ;
 LuniDeToamna=(Septembrie..Noiembrie) ;

⑤ Se consideră următorul program:

```
Program P21;  
type Indice=1..10;  
var i, j, k, m : Indice;  
begin  
  writeln('Introduceți indicii i, j:');  
  readln(i, j);  
  k:=i+j; writeln('k=', k);  
  m:=i-j; writeln('m=', m);  
end.
```

Pentru care valori ale variabilelor i, j se vor declanșa erori de execuție?

- | | |
|--------------|---------------|
| a) i=3, j=2; | e) i=2, j=2; |
| b) i=7, j=4; | f) i=3, j=11; |
| c) i=4, j=7; | g) i=8, j=4; |
| d) i=6, j=3; | h) i=5, j=3. |

⑥ Se consideră programul P20. După lansarea în execuție utilizatorul introduce x=1, y=2. Evident, x-y=-1. Întrucât valoarea -1 nu aparține tipului de date Pozitiv, la execuția instrucțiunii

z:=x-y

va surveni o eroare.

Indicați instrucțiunile la execuția cărora se vor declanșa erori, dacă:

- | | |
|----------------------|-------------------|
| a) x=1000, y=1000; | e) x=1, y=2; |
| b) x=1000, y=1001; | f) x=1000, y=100; |
| c) x=1001, y=1000; | g) x=0, y=1; |
| d) x=30000, y=30000; | h) x=1, y=0. |

2.8. GENERALITĂȚI DESPRE TIPURILE ORDINALE DE DATE

Tipurile de date integer, boolean, char, *enumerare* și *subdomeniu* se numesc **tipuri ordinale**. Fiecare valoare a unui tip ordinal are un număr de ordine, definit după cum urmează:

1) numărul de ordine al unui număr de tip integer este însuși numărul considerat;

2) valorile de adevăr false și true ale tipului boolean au numerele de ordine, respectiv, 0 și 1;

3) numărul de ordine al unui caracter (tipul char) este dat de poziția lui în tabelul de codificare, obișnuit *ASCII*;

4) numărul de ordine al unei valori de tip *enumerare* este dat de poziția ei în lista de enumerare. De remarcat că valorile unei liste sînt numerotate prin 0, 1, 2, ... ș.a.m.d.;

5) valorile unui tip *subdomeniu* moștenesc numerele de ordine de la tipul de bază.

Numărul de ordine al unei valori de tip ordinal poate fi aflat cu ajutorul funcției predefinite `ord`.

Programul P22 afișează pe ecran numerele de ordine ale valorilor `-32`, `true`, `'A'`, `A` și `B`.

```
Program P22;  
{ Numerele de ordine ale valorilor de tip ordinal }  
type T1=(A, B, C, D, E, F, G, H);  
      T2=B..G;  
begin  
  writeln(ord(-32)); { -32 }  
  writeln(ord(true)); { 1 }  
  writeln(ord('A')); { 65 }  
  writeln(ord(A)); { 0 }  
  writeln(ord(B)); { 1 }  
end.
```

Asupra valorilor oricărui tip ordinal de date sînt permise operațiile relaționale cunoscute:

< *mai mic;*
<= *mai mic sau egal;*
= *egal;*
>= *mai mare sau egal;*
> *mai mare;*
<> *diferit.*

Rezultatul unei operații relaționale este de tip boolean, `false` sau `true`, în funcție de numerele de ordine ale valorilor operanzilor.

De exemplu, în prezența declarațiilor

```
type Culoare=(Galben, Verde, Albastru, Violet);
```

rezultatul operației

```
Verde<Violet
```

este `true`, deoarece `ord(Verde)=1`, `ord(Violet)=3` și 1 este mai mic ca 3.

Rezultatul operației

```
Galben>Violet
```

este `false`, deoarece `ord(Galben)=0`, `ord(Violet)=3` și 0 nu este mai mare ca 3.

Programul ce urmează afișează pe ecran rezultatele operațiilor relaționale pentru valorile `Verde` și `Violet` ale tipului de date `Culoare`.

```
Program P23;  
{Operații relaționale asupra valorilor
```

```

de tip ordinal }
type Culoare=(Galben, Verde, Albastru, Violet);
begin
  writeln(Verde<Violet); { true }
  writeln(Verde<=Violet); { true }
  writeln(Verde=Violet); { false }
  writeln(Verde>=Violet); { false }
  writeln(Verde>Violet); { false }
  writeln(Verde<>Violet); { true }
end.

```

Pentru tipurile ordinale de date există funcțiile predefinite *pred* (*predecesor*) și *succ* (*succesor*).

Predecesorul valorii ordinale cu numărul de ordine i este valoarea cu numărul de ordine $i - 1$. *Succesorul* valorii ordinale în studiu este valoarea cu numărul de ordine $i + 1$.

De exemplu, pentru valorile tipului ordinal de date Culoare obținem:

```

pred(Verde)= Galben;
succ(Verde)= Albastru;
pred(Albastru)= Verde;
succ(Albastru)= Violet.

```

Evident, valoarea cea mai mică nu are predecesor, iar valoarea cea mai mare nu are succesor.

Programul P24 afișează pe ecran predecesorii și succesorii valorilor ordinale 'B', 0 și '0'.

```

Program P24;
{ Predecesorii și succesorii valorilor ordinale }
begin
  writeln(pred('B')); { 'A' }
  writeln(succ('B')); { 'C' }
  writeln(pred(0)); { -1 }
  writeln(succ(0)); { 1 }
  writeln(pred('0')); { '/' }
  writeln(succ('0')); { '1' }
end.

```

Se observă că predecesorii și succesorii valorilor ordinale 0 (tip integer) și '0' (tip char) nu coincid, deoarece tipurile valorilor respective diferă.

Subliniem faptul că tipul de date real nu este un tip ordinal. Prin urmare, asupra valorilor reale nu pot fi aplicate funcțiile ord (numărul de ordine), pred (predecesor) și succ (succesor). Nerespectarea acestei reguli va declanșa erori.

Întrebări și exerciții

- ❶ Numiți tipurile ordinale de date. Care sînt proprietățile lor comune?
- ❷ Cum se definesc numerele de ordine ale valorilor unui tip ordinal de date?
- ❸ Ce va afișa pe ecran programul ce urmează?

```
Program P25;  
type Zi=(L, Ma, Mi, J, V, S, D);  
var z1, z2 : Zi;  
begin  
    z1:=Ma;  
    writeln(ord(z1));  
    z2:=pred(z1);  
    writeln(ord(z2));  
    z2:=succ(z1);  
    writeln(ord(z2));  
    z1:=Mi; z2:=V;  
    writeln(z1<z2);  
    writeln(z1>z2);  
    writeln(z1<>z2);  
end.
```

- ❹ Exclueți din programul de mai jos linia care conține o eroare:

```
Program P26;  
{ Eroare }  
var i : integer;  
begin  
    i:=MaxInt;  
    writeln(pred(i));  
    writeln(succ(i));  
end.
```

Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

- ❺ Comentați programul ce urmează:

```
Program P27;  
{ Eroare }  
var i : integer; x : real;  
begin  
    i:=1; x:=1.0;  
    writeln(ord(i));  
    writeln(ord(x));  
    writeln(pred(i));  
    writeln(pred(x));  
    writeln(succ(i));  
    writeln(succ(x));  
end.
```

Exclueți liniile care conțin erori. Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

2.9. DEFINIREA TIPURILOR DE DATE

Limbajul PASCAL oferă utilizatorului tipurile predefinite de date integer, real, boolean, char ș.a. Dacă e necesar, programatorul poate defini tipuri proprii de date, de exemplu, *enumerare* și *subdomeniu*.

Denumirea unui tip de date și mulțimea lui de valori se definesc cu ajutorul următoarelor unități gramaticale:

```
<Tipuri> ::= type <Definiție tip>; { <Definiție tip>; }  
<Definiție tip> ::= <Identificator> = <Tip>  
<Tip> ::= <Identificator> | <Tip enumerare> |  
           <Tip subdomeniu> | <Tip tablou> |  
           <Tip articol> | <Tip mulțime> |  
           <Tip fișier> | <Tip referință>  
<Tip enumerare> ::= (<Identificator> {, <Identificator>})  
<Tip subdomeniu> ::= <Constantă> .. <Constantă>
```

Diagramele sintactice corespunzătoare sînt prezentate în *fig. 2.2*.

Exemple:

```
1) type    T1=(A, B, C, D, E, F, G, H);  
           T2=B..F;  
           T3=C..H;  
2) type    Pozitiv=1..MaxInt;  
           Natural=0..MaxInt;  
           Negativ=-MaxInt..-1;  
3) type    Abatere=-10...+10;  
           Litera='A'..'Z';  
           Cifra='0'..'9';
```

Clasificarea tipurilor de date ale limbajului PASCAL este prezentată în *fig. 2.3*. Tipurile studiate deja sînt prezentate pe un fundal gri.

În anumite construcții ale limbajului PASCAL variabilele și constantele trebuie să fie de tipuri identice sau compatibile.

Două tipuri sînt **identice**, dacă ele au fost definite cu același nume de tip.

De exemplu, fie

```
type    T4=integer;  
           T5=integer;
```

Aici tipurile integer, T4 și T5 sînt identice.

Două tipuri sînt identice și atunci cînd sînt definite cu nume diferite, dar aceste nume sînt echivalente prin tranzitivitate.

De exemplu, fie

```
type    T6=real;  
           T7=T6;  
           T8=T7;
```

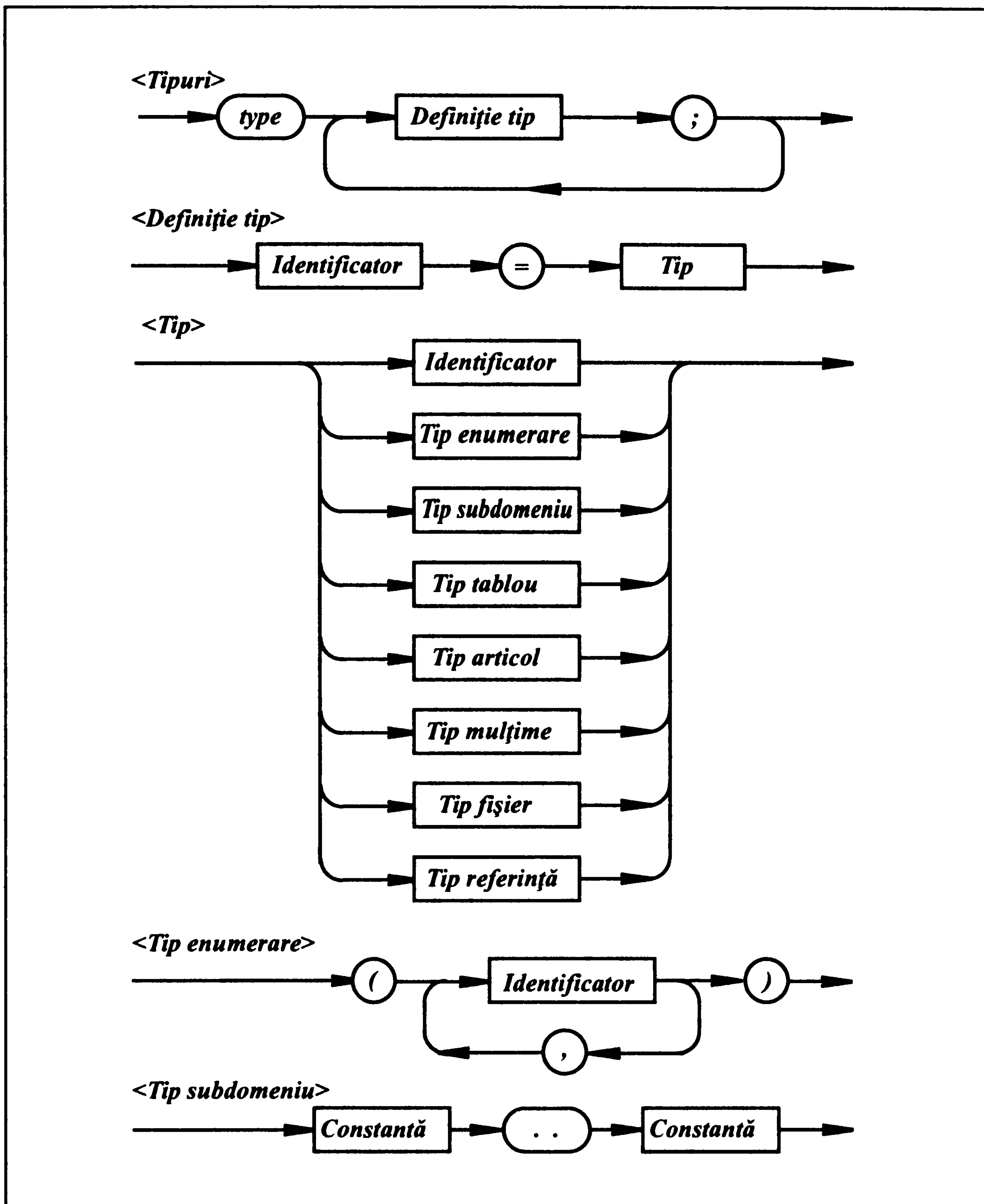


Fig. 2.2. Diagramele sintactice pentru definirea tipurilor de date

Aici *real*, T6, T7 și T8 sînt tipuri identice.

Două tipuri sînt **compatibile** atunci cînd este adevărată cel puțin una din următoarele afirmații:

- cele două tipuri sînt identice;
- un tip este un subdomeniu al celui alt tip;
- ambele tipuri sînt subdomenii ale aceluiași tip de bază.

De exemplu, în prezența declarațiilor

Type Zi=(L, Ma, Mi, J, V, S, D);
 ZiDeLucru=(L, Ma, Mi, J, V);
 ZiDeOdihna=(S, D);
 Culoare=(Galben, Verde, Albastru, Violet);

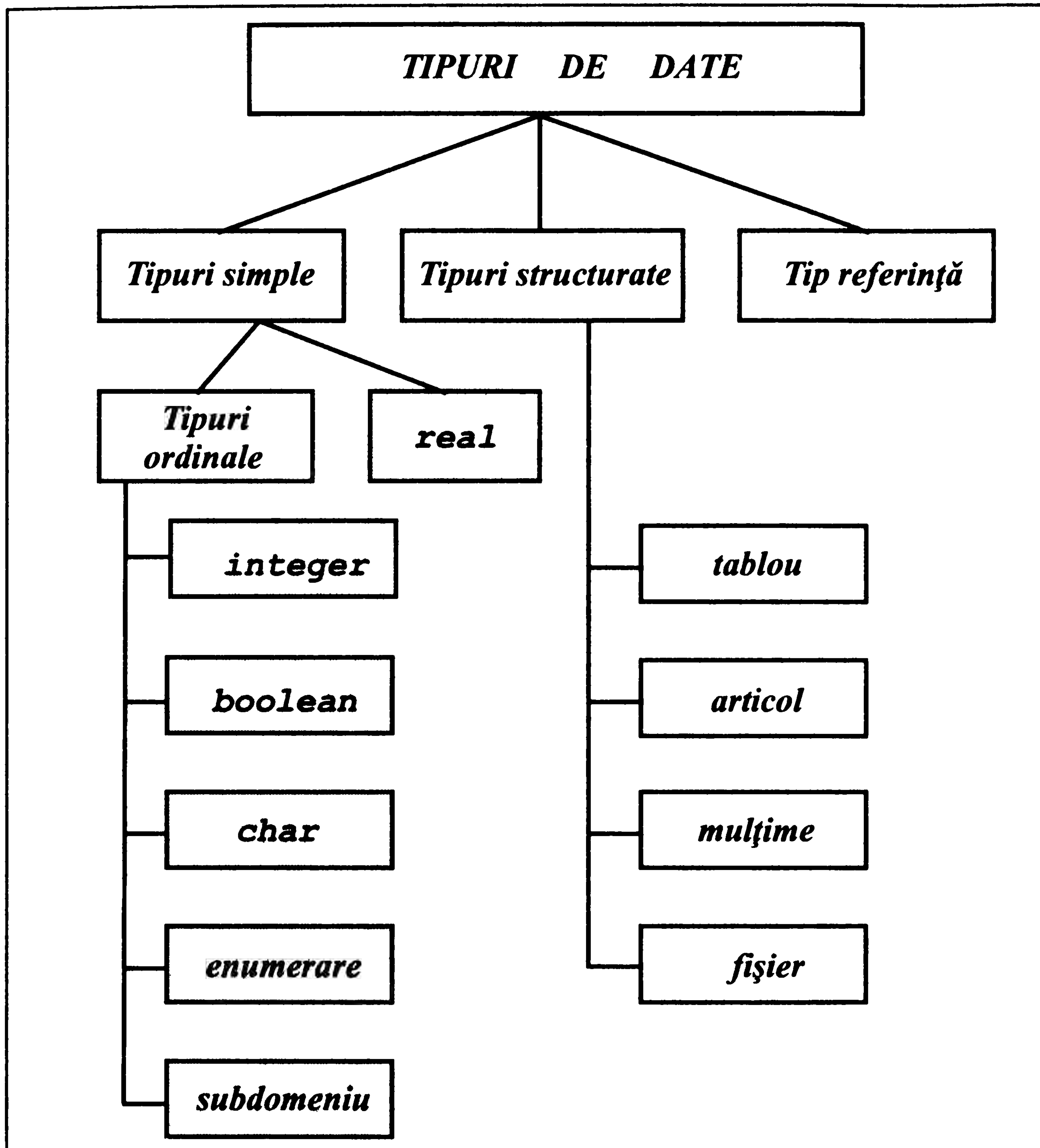


Fig. 2.3. Clasificarea tipurilor de date

tipurile Zi, ZiDeLucru, ZiDeOdihna sînt compatibile. Tipurile Zi și Culoare sînt tipuri incompatibile. Prin urmare, sînt admise operațiile relaționale

L<D
 Mi<>D
 Verde<>Violet

etc. și nu sînt admise operațiile de tipul:


```
L<Violet  
Verde=V  
S<>Albastru
```

ș.a.m.d.

În completare la tipurile de date definite de utilizator explicit cu ajutorul cuvîntului-cheie **type**, într-un program PASCAL pot fi definite și tipuri anonime (fără denumire).

Un **tip anonim** se definește implicit într-o declarație de variabile.

Exemplu:

```
var i : 1..20;  
    s : (Alfa, Beta, Gama, Delta);  
    t : Alfa..Gama;
```

Se observă că tipul subdomeniu 1..20, tipul *enumerare* (Alfa, Beta, Gama, Delta) și tipul *subdomeniu* Alfa..Gama nu au denumiri proprii.

De regulă, tipurile anonime se utilizează în programele cu un număr mic de variabile.

Întrebări și exerciții

❶ Se consideră următorul program:

```
Program P28;  
type T1=-100..100;  
      T2='A'..'H';  
      T3=(A, B, C, D, E, F, G, H);  
      T4=A..E;  
      T5=integer;  
      T6=real;  
      T7=char;  
      T8=boolean;  
var i : T1;    j : T5;  
    k : T2;    m : T3;  
    n : T4;    p : real;  
    q : T6;    r : char;  
    s : T7;    t : boolean;  
    z : T8;    y : real;  
begin  
    { ... calcule ce utilizează }  
    { variabilele în studiu ... }  
    writeln('Sfîrșit');  
end.
```

Precizați tipurile de date ale programului. Ce valori poate lua fiecare variabilă din acest program? Care tipuri sînt compatibile?

- ② Indicați pe diagramele sintactice din *fig. 2.2* drumurile care corespund definițiilor tipurilor de date din programul P28.
- ③ Precizați tipurile anonime de date din următorul program:

```
Program P29;  
  type T1=integer;  
        T2=-150..150;  
        T3=1..5;  
  var i : T1;  
      j : T2;  
      k : T3;  
      m : 1..5;  
      n : (Unu, Doi, Trei, Patru, Cinci);  
begin  
  { ... calcule ce utilizează }  
  { variabilele în studiu ... }  
  writeln('Sfârșit')  
end.
```

Ce valori poate lua fiecare variabilă din acest program?

- ④ Se consideră următoarele declarații:

```
type T1=boolean;  
      T2=T1;  
      T3=T2;  
      T4=T3;  
var x:T4;
```

Precizați valorile pe care le poate lua variabila *x* și operațiile tipului corespunzător de date.

- ⑤ Când două tipuri de date sînt identice? Dați exemple.
- ⑥ Când două tipuri de date sînt compatibile? Dați exemple.
- ⑦ Se consideră declarațiile:

```
type T1=integer;  
      T2=T1;  
      T3=-5..+5;  
      T4=T3;  
      T5=-10..+10;  
      T6=(A,B,C,D,E,F,G,H);  
      T7=A..D;  
      T8=E..H;  
      T9='A'..'D';  
      T10='E'..'H';
```

Precizați tipurile identice și tipurile compatibile de date.

2.10. DECLARAȚII DE VARIABILE

Cunoaștem deja că fiecare variabilă care apare într-un program PASCAL în mod obligatoriu se asociază cu un anumit tip de date. Pentru aceasta se utilizează următoarele construcții gramaticale:

$\langle \text{Variabile} \rangle ::=$
var $\langle \text{Declarație variabile} \rangle$; { $\langle \text{Declarație variabile} \rangle$; }
 $\langle \text{Declarație variabile} \rangle ::=$
 $\langle \text{Identificator} \rangle$ { , $\langle \text{Identificator} \rangle$ } : $\langle \text{Tip} \rangle$

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *fig. 2.4*.

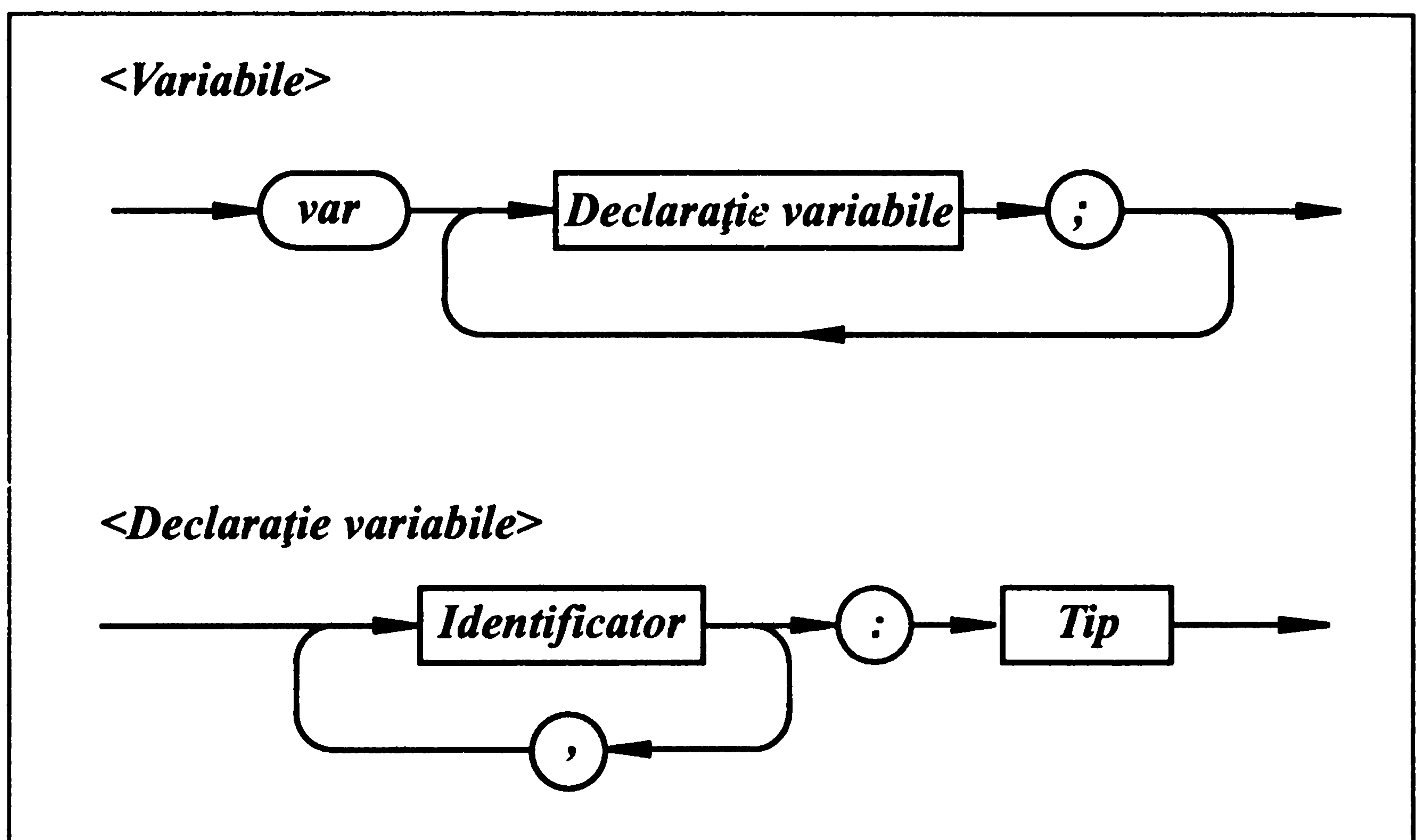


Fig. 2.4. Diagramele sintactice <Variabile> și <Declarație variabile>

Declarațiile de variabile pot utiliza tipuri predefinite de date (integer, real, char, boolean ș.a.) și tipuri definite de utilizator (enumerare, subdomeniu etc.).

Exemple:

```
1) var   i, j : integer;  
        x : real;  
        p : boolean;  
        r, s : char;  
2) type T1=(A, B, C, D, E, F, G, H);  
        T2=C..F;  
        T3=1..10;  
        T4='A'...'Z';
```



```

    var x, y, z : real;
        r, s : char;
        i, j : integer;
        k : T3;
        p : T1;
        q : T2;
        v : T4;
3) type Zi=(L, Ma, Mi, J, V, S, D);
    var x, y : real;
        z : Zi;
        z1 : L..V;
        m, n : 1..10;

```

Menționăm că în ultimul exemplu tipul variabilelor *z1*, *m* și *n* este definit direct în declarațiile de variabile. Prin urmare, variabilele în studiu aparțin unor tipuri anonime de date.

Întrebări și exerciții

❶ Determinați tipul variabilelor din următorul program:

```

Program P30;
type T1=integer;
    Studii=(Elementare, Medii, Superioare);
    T2=real;
    Culoare=(Galben, Verde, Albastru, Violet);
var x : real;
    y : T1;
    i : integer;
    j : T2;
    p : boolean;
    c : Culoare;
    s : Studii;
    q : Galben..Albastru;
    r : 1..9;
begin
    { ... calcule în care se utilizează }
    { variabilele în studiu ... }
    writeln('Sfîrșit');
end.

```

Precizați valorile pe care le poate lua fiecare variabilă și operațiile tipului corespunzător de date.

- ❷ Indicați pe diagramele sintactice din *fig. 2.4* drumurile care corespund declarațiilor de variabile din programul P30.
- ❸ Ce mesaje vor fi afișate pe ecran pe parcursul compilării următorului program?

```

Program P31;
var i, j : integer;
begin

```

```

i:=1; j:=2; k:=i+j;
  writeln('k=', k);
end.

```

- ④ Cum se declară variabilele unui tip anonim de date?

2.11. DEFINIȚII DE CONSTANTE

Se știe că valorile unui tip de date pot fi referite prin variabile și constante. Pentru a face programele mai lizibile și ușor de modificat, limbajul PASCAL permite reprezentarea constantelor prin denumiri simbolice. Identificatorul care reprezintă o constantă se numește *nume de constantă* sau, pur și simplu, *constantă*. Peste tot în program, unde apare un astfel de nume, el va fi înlocuit cu valoarea corespunzătoare.

Definirea numelor de constante se face cu ajutorul următoarelor construcții gramaticale:

```

<Constante> ::=
    const <Definiție constantă>; { <Definiție constantă>; }
<Definiție constantă> ::= <Identificator> = <Constantă>
<Constantă> ::=
    [+ | -] <Număr, fără semn> |
    [+ | -] <Nume de constantă> |
    <Șir de caractere>

```

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *fig. 2.5*.

Exemple:

- 1) **const** a=10;
 b=9.81;
 c='*';
 t='TEXT';
- 2) **const** NumarCaractere=60;
 LungimePagina=40;
- 3) **const** n=10;
 m=20;
 Pmax=2.15e+8;
 Pmin=-Pmax;
 S='STOP';

Spre deosebire de variabile, tipul cărora se indică explicit în declarațiile de variabile, tipul constantelor se indică implicit prin forma lor textuală. În exemplele de mai sus tipul constantelor este:

a, NumarCaractere, LungimePagina, n, m — integer;
 b, Pmax, Pmin — real;

c — char;
t, S — șir de caractere.

În programul P32 se definesc constantele Nmax, Nmin, Pi, Separator, Indicator și Mesaj. Valorile acestor constante se afișează pe ecran.

```
Program P32;  
{ Definiții de constante }  
  const Nmax=40;      { Constantă de tip integer }  
        Nmin=-Nmax;   { Constantă de tip integer }  
        Pi=3.14;      { Constantă de tip real }  
        Separator='\\'; { Constantă de tip char }  
        Indicator=TRUE; { Constantă de tip boolean }  
        Mesaj='Verificați imprimanta';  
                { Șir de caractere }  
  
begin  
  writeln(Nmax);  
  writeln(Nmin);  
  writeln(Pi);  
  writeln(Separator);  
  writeln(Indicator);  
  writeln(Mesaj);  
  { ... calcule în care se utilizează }  
  { constante în studiu ... }  
end.
```

În mod obișnuit, datele constante ale unui program, de exemplu, numărul de linii ale unui tabel, numărul π , accelerația căderii libere g ș.a.m.d., se includ în definiții de constante. Acest fapt permite modificarea constantelor, fără a schimba programul în întregime.

În programul P33 lungimea L și aria cercului S se calculează conform formulelor:

$$L = 2\pi r ;$$
$$S = \pi r^2 ,$$

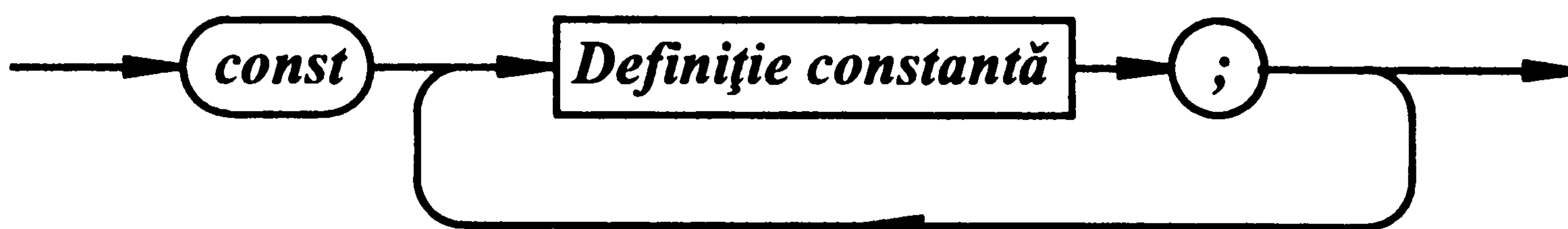
unde r este raza cercului. Numărul π este reprezentat prin constanta $Pi = 3.14$.

```
Program P33;  
{ Lungimea și aria cercului }  
  const Pi=3.14;  
  var L, S , r :real;  
  begin  
    writeln('Introduceți raza r:');  
    readln(r);  
    L:=2*Pi*r;  
    writeln('Lungimea L=', L);  
    S:=Pi*r*r;
```

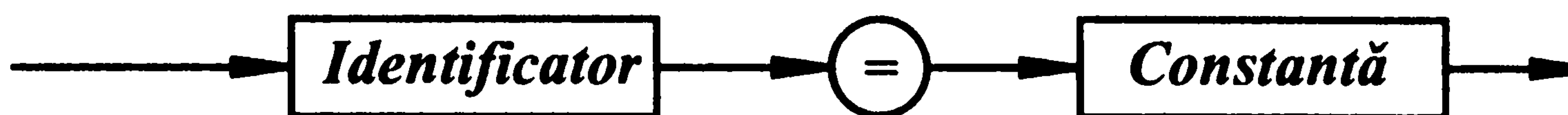


```
writeln('Aria S=', S);
end.
```

<Constante>



<Definiție constantă>



<Constantă>

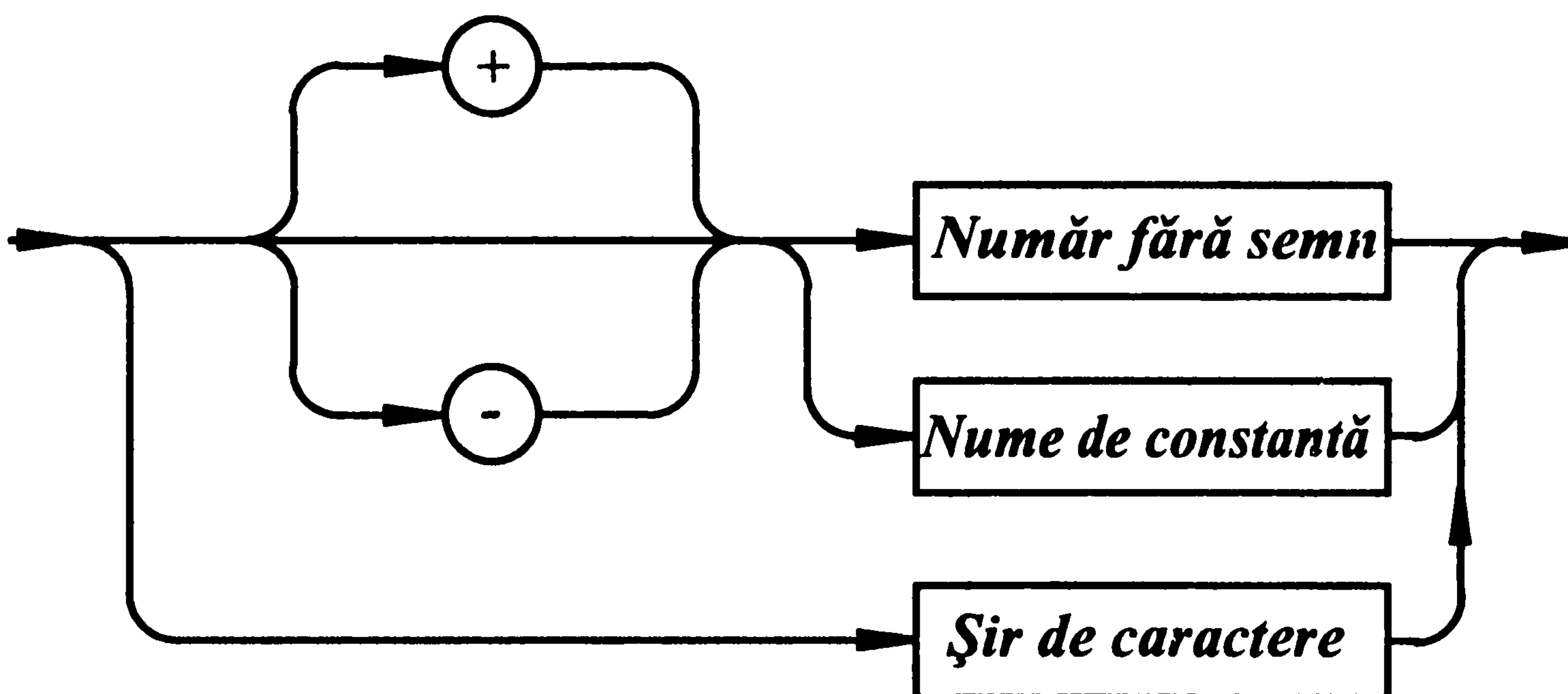


Fig. 2.5. Diagramele sintactice **<Constante>**, **<Definiție constantă>** și **<Constantă>**

Dacă utilizatorul are nevoie de rezultate mai exacte, se modifică numai linia a treia a programului P33:

```
const Pi=3.141592654;
```

Evident, restul programului rămîne neschimbat.

Spre deosebire de variabile, valorile constantelor nu pot fi modificate prin atribuire sau operații de citire. Nerespectarea acestei reguli va declanșa erori de compilare.

Întrebări și exerciții

❶ Determinați tipul următoarelor constante:

a) const a=29.1;	b) const d=-16.82e-14;
b=TRUE;	f=-d;
c=18;	i=15;
c) const t='F';	d) const x=65;
q=FALSE;	y=-x;
m='PAUZĂ';	z=-y;

❷ Elaborați un program care afișează pe ecran valorile constantelor din exercițiul 1.

❸ Indicați pe diagramele sintactice din *fig. 2.5* drumurile care corespund definițiilor de constante din programul P32.

❹ Inserați între cuvintele-cheie **begin** și **end** ale programului P32 una dintre liniile ce urmează:

```
    Nmax:=10;  
    readln(Nmax);
```

Explicați mesajele afișate pe ecran pe parcursul compilării programului modificat.

❺ Se consideră programul:

```
Program P34;  
  const t='1'; s=-t;  
  begin  
    writeln(s);  
  end.
```

Ce mesaje vor fi afișate pe ecran în procesul compilării?

❻ Ce rezultate se vor afișa după execuția următorului program:

```
Program P35;  
  const i=1; j=i; k=j;  
  var x, y, z : integer;  
  begin  
    x:=i+1; writeln(x);  
    y:=j+2; writeln(y);  
    z:=k+3; writeln(z);  
  end.
```

❼ Se consideră programul:

```
Program P36;  
  const a=1; b=2; c=3; d=4;  
  var i, j, k : integer;
```

```
begin  
  i:=a+1; writeln(i);  
  j:=b+1; writeln(j);  
  k:=c+1; writeln(k);  
  d:=a+1; writeln(d);  
end.
```

Explicați mesajele afișate pe ecran.

- ⑧ Exclueți din programul ce urmează linia care conține o eroare.

```
Program P37;  
const a=1;  
var i, j : integer;  
begin  
  writeln('Introduceți i=');  
  readln(i); j:=i+a;  
  writeln(j);  
  writeln('Introduceți a=');  
  readln(a);  
  j:=i+a;  
  writeln(j);  
end.
```

Ce rezultate vor fi afișate pe ecran după execuția programului modificat?

Capitolul 3

.....

INSTRUCȚIUNI

3.1. CONCEPTUL DE ACȚIUNE

Un program PASCAL constă din două părți: partea declarativă și partea executabilă. În partea declarativă se descriu datele cu care va opera programul, iar în partea executabilă se descriu acțiunile (operațiile) cu aceste date.

Acțiunile necesare pentru a prelucra datele unui program și ordinea executării lor se definesc cu ajutorul **instrucțiunilor**. Există două categorii de instrucțiuni:

- 1) instrucțiuni simple;
- 2) instrucțiuni structurate.

Instrucțiunile simple nu conțin alte instrucțiuni. Instrucțiunile simple sînt:

- instrucțiunea de atribuire;
- instrucțiunea de apel de procedură;
- instrucțiunea de salt necondiționat;
- instrucțiunea de efect nul.

Instrucțiunile structurate sînt construite din alte instrucțiuni. Instrucțiunile structurate sînt:

- instrucțiunea compusă;
- instrucțiunile condiționale **if** și **case**;
- instrucțiunile iterative **for**, **while** și **repeat**;
- instrucțiunea **with**.

În cadrul unui program instrucțiunile pot fi prefixate de etichete. Etichetele pot fi referite în instrucțiunile de salt necondiționat **goto**. Amintim că eticheta este un număr întreg fără semn (vezi paragraful 1.10).

Diagrama sintactică *<Instrucțiune>* este prezentată în *fig. 3.1*. Menționăm că eticheta se separă de instrucțiunea propriu-zisă prin simbolul “:” (două puncte).

Într-un program PASCAL scrierea instrucțiunilor pe linii nu este limitată, o instrucțiune poate ocupa una sau mai multe linii, sau într-o linie pot fi mai multe instrucțiuni. Ca separator de instrucțiuni se folosește simbolul “;” (punct și virgulă).

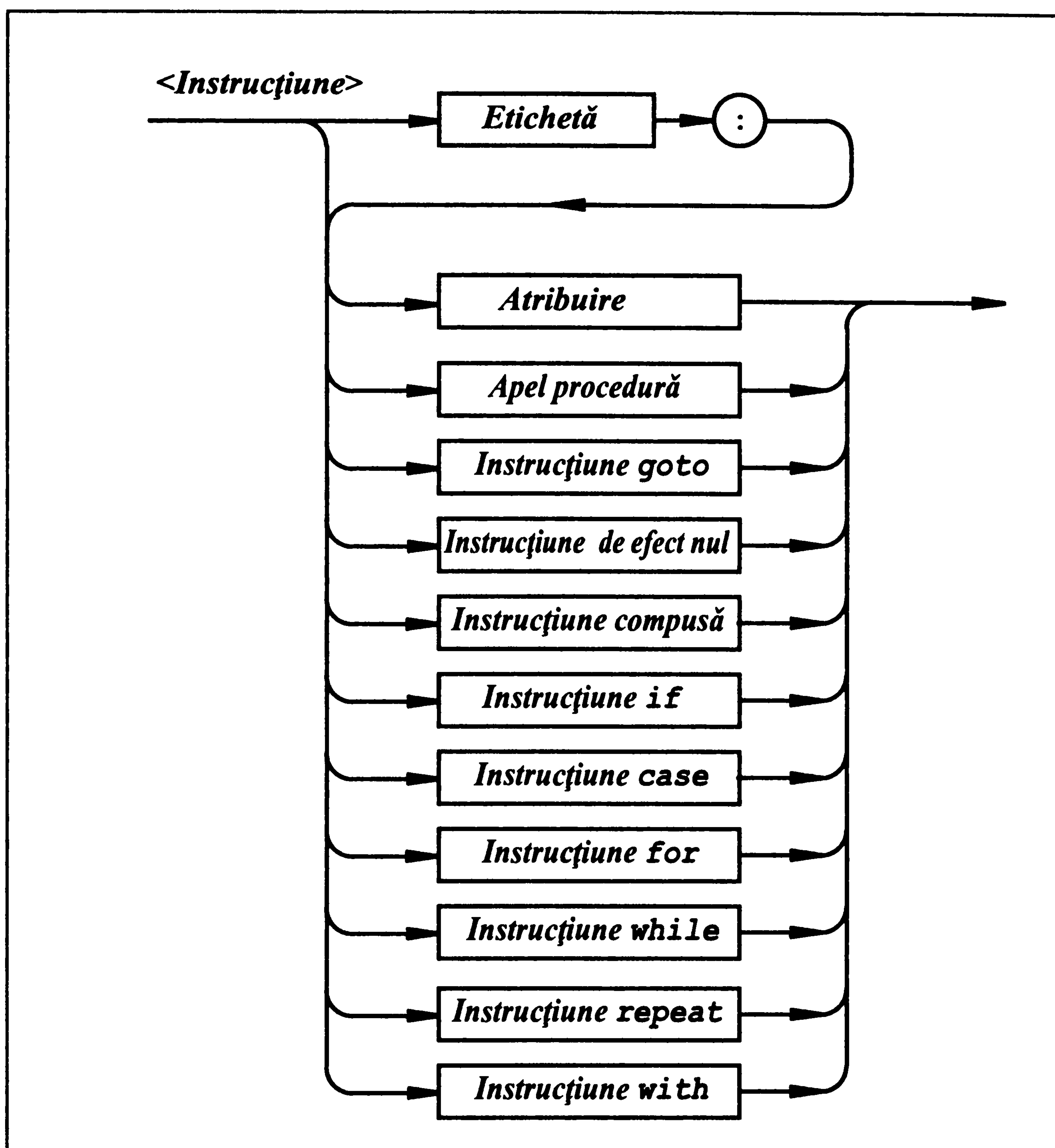


Fig. 3.1. Diagrama sintactică <Instrucțiune>

3.2. EXPRESII

Formulele pentru calculul unor valori se reprezintă în PASCAL prin **expresii**. Acestea sînt formate din operanzi (constante, variabile, referințe de funcții) și operatori (simbolurile operațiilor). Operatorii se clasifică după cum urmează:

<Operator multiplicativ> ::= * | / | **div** | **mod** | **and**

<Operator aditiv> ::= + | - | **or**

<Operator relațional> ::= < | <= | = | >= | > | <> | **in**

În componența expresiilor intră factori, termeni și expresii simple.

Factorul poate fi o variabilă, o constantă fără semn, apelul unei funcții ș.a. Mai exact,

$\langle \textit{Factor} \rangle ::= \langle \textit{Variabilă} \rangle \mid \langle \textit{Constantă fără semn} \rangle \mid \langle \textit{Apel funcție} \rangle \mid \mathbf{not} \langle \textit{Factor} \rangle \mid (\langle \textit{Expresie} \rangle) \mid \langle \textit{Constructor mulțime} \rangle$

Exemple:

15
x
p
sin(x)
not p

Un **termen** are forma:

$\langle \textit{Termen} \rangle ::= \langle \textit{Factor} \rangle \{ \langle \textit{Operator multiplicativ} \rangle \langle \textit{Factor} \rangle \}$

Exemple:

15
x
15* x
x* y* z
p **and** q
sin(x) / 3

Prin **expresie simplă** se înțelege:

$\langle \textit{Expresie simplă} \rangle ::= [+ \mid -] \langle \textit{Termen} \rangle \{ \langle \textit{Operator aditiv} \rangle \langle \textit{Termen} \rangle \}$

Exemple:

+15
-x
15* x+sin(x) / 3
p **or** q

La rîndul său, o **expresie** are forma:

$\langle \textit{Expresie} \rangle ::= \langle \textit{Expresie simplă} \rangle \{ \langle \textit{Operator relațional} \rangle \langle \textit{Expresie simplă} \rangle \}$

Exemple:

-15
x* y+cos(z) / 4
x<15
15* x+sin(x) / 3<>11
x+6>z-3.0

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în fig. 3.2 și 3.3.

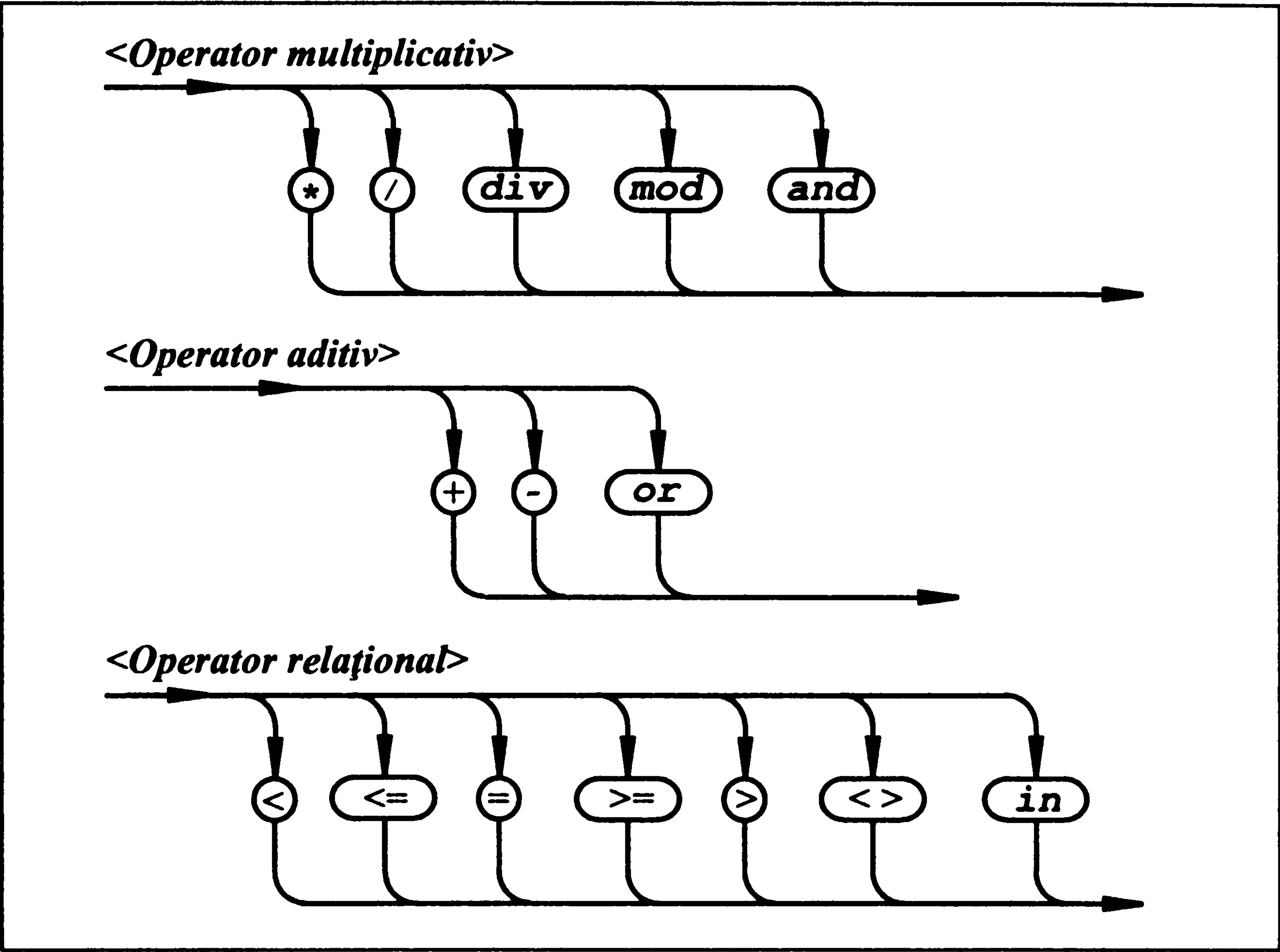


Fig. 3.2. Diagramele sintactice pentru operatori

O expresie luată în paranteze se transformă într-un factor. Cu astfel de factori se pot forma noi termeni, expresii simple, expresii ș.a.m.d.

Exemple:

<u>notația matematică</u>	<u>notația în PASCAL</u>
$\frac{a + b}{c + d}$	(a+b) / (c+d)
$\frac{-b + \sin(b - c)}{2 a}$	(-b+sin (b-c)) / (2* a)
$-\frac{1}{x y}$	-1 / (x* y)
$\underline{p < q} \ \& \ r > s$	(p<q) and (r>s)
$x \vee y$	not (x or y)
$\frac{1}{a + b} > \frac{1}{c + d}$	1 / (a+b)>1 / (c+d)

În cadrul unei expresii un apel de funcție poate să apară peste tot unde apare o variabilă sau o constantă (fig. 3.3). Limbajul PASCAL conține un set de **funcții predefinite**, cunoscute oricărui program. Aceste funcții sînt prezentate în *tabelul 3.1*.

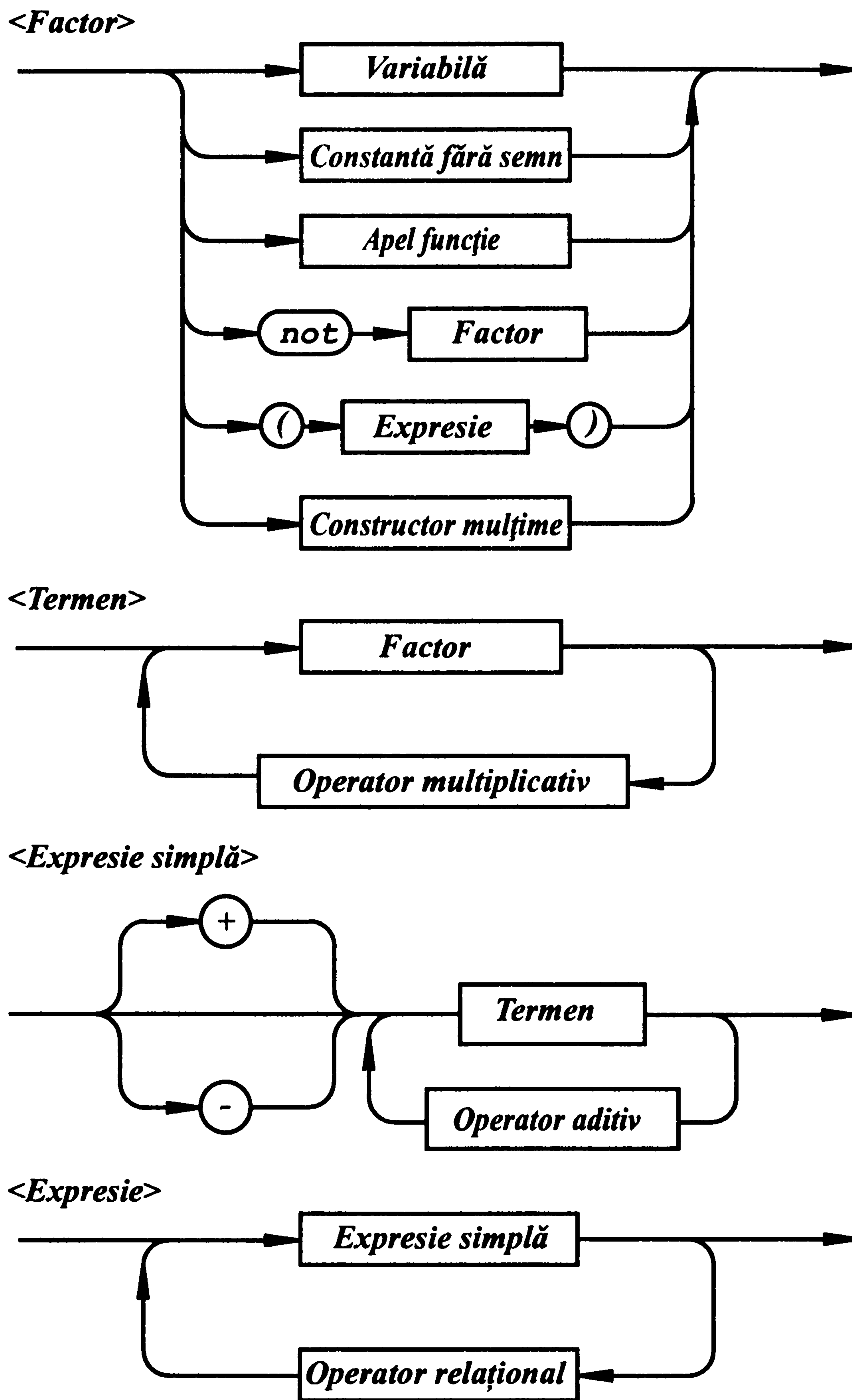


Fig. 3.3. Diagramele sintactice pentru definirea expresiilor

Tabelul 3.1. Funcțiile predefinite ale limbajului PASCAL

DENUMIREA FUNCȚIEI	NOTAȚIA ÎN PASCAL
valoarea absolută $ x $	<code>abs (x)</code>
sinus $\sin x$	<code>sin (x)</code>
cosinus $\cos x$	<code>cos (x)</code>
arctangenta $\arctg x$	<code>arctan (x)</code>
pătratul lui x x^2	<code>sqr (x)</code>
rădăcina pătrată \sqrt{x}	<code>sqrt (x)</code>
puterea numărului e e^x	<code>exp (x)</code>
logaritmul natural $\ln x$	<code>ln (x)</code>
rotunjirea lui x	<code>round (x)</code>
trunchierea lui x	<code>trunc (x)</code>
paritatea numărului i (<i>false</i> pentru i par și <i>true</i> în caz contrar)	<code>odd (i)</code>
numărul valorii ordinale v	<code>ord (v)</code>
predecesorul lui v	<code>pred (v)</code>
succesorul lui v	<code>succ (v)</code>
caracterul cu numărul i	<code>chr (i)</code>
testarea sfârșitului de fișier	<code>eof (f)</code>
testarea sfârșitului de linie	<code>eoln (f)</code>

Întrebări și exerciții

❶ Scrieți conform regulilor limbajului PASCAL următoarele expresii:

- a) $a^2 + b^2$;

b) $a^2 + 2ab + b^2$;

c) $(a + b)^2$;

d) $v_0 t + \frac{a t^2}{2}$;

e) $\frac{-b + \sqrt{b^2 - 4ac}}{2a}$;

f) $\cos \alpha + \cos \beta$;

g) $\cos(\alpha + \beta)$;
- h) $2\pi r$;

i) πr^2 ;

j) $x_1 x_2 \vee x_3 x_4$;

k) $\overline{x_1 \vee x_2}$;

l) $|x| < 3$;

m) $z < 6 \& |q| > 13,4$;

n) $x > 0 \& y > 8 \& R < 15$.

❷ Indicați pe diagramele sintactice din fig. 3 3 drumurile care corespund următoarelor expresii PASCAL:

- a) `x`

b) `3 . 14`

c) `sin (x)`

d) `not q`

- h) $(a > b)$ **and** $(c > d)$

3 Transpuneți expresiile PASCAL în notații obișnuite:

- ### h) not (x and y)

④ Care din expresiile PASCAL ce urmează sînt greșite? Pentru a argumenta răspunsul, utilizați diagramele sintactice din *fig. 3.3*.

- ```
n) sqrt(-y)
```

### 3.3. EVALUAREA EXPRESIILOR

Prin evaluarea unei expresii se înțelege calculul valorii ei. Rezultatul furnizat depinde de valorile operanzilor și de operatorii care acționează asupra acestora.

Regulile de evaluare a unei expresii sînt cele obișnuite în matematică:

- mai întâi se calculează expresiile dintre paranteze.

Prioritățile operatorilor sînt indicate în *tabelul 3.2*.

***Tabelul 3.2. Prioritățile operațiilor limbajului PASCAL***

| <i>CATEGORIE</i>         | <i>OPERATORI</i>           | <i>PRIORITATE</i>      |
|--------------------------|----------------------------|------------------------|
| operatori unari          | not, @                     | prima (cea mai mare)   |
| operatori multiplicativi | *, /, div,<br>mod, and     | a doua                 |
| operatori aditivi        | +, -, or                   | a treia                |
| operatori relaționali    | <, <=, =, >=,<br>>, <>, in | a patra (cea mai mică) |

### Exemplu:

Fie  $x=2$  și  $y=6$ . Atunci:

a)  $2 * x + y = 2 \cdot 2 + 6 = 4 + 6 = 10$ ;

b)  $2 * (x + y) = 2 \cdot (2 + 6) = 2 \cdot 8 = 16$ ;

c)  $x + y / x - y = 2 + 6 / 2 - 6 = 2 + 3 - 6 = 5 - 6 = -1$ ;

d)  $(x + y) / x - y = (2 + 6) / 2 - 6 = 8 / 2 - 6 = 4 - 6 = -2$ ;

e)  $x + y / (x - y) = 2 + 6 / (2 - 6) = 2 + 6 / (-4) = 2 + (-1,5) = 0,5$ ;

f)  $x + y < 15 = 2 + 6 < 15 = 8 < 15 = \text{true}$ ;

g)  $(x + y < 15) \text{ and } (x > 3) = (2 + 6 < 15) \text{ and } (2 > 3) = (8 < 15) \text{ and } 2 > 3 = \text{true and false} = \text{false}$ .

Se observă că părțile componente ale unei expresii (*fig. 3.3*) se calculează în următoarea ordine:

- |              |                           |
|--------------|---------------------------|
| 1) factorii; | 3) expresiile simple;     |
| 2) termenii; | 4) expresia propriu-zisă. |

Valoarea curentă a unei expresii poate fi afișată pe ecran cu ajutorul procedurii `writeln`:

`writeln(<Expresie>)`

Programul P38 afișează pe ecran rezultatele evaluării expresiilor  $x * y + z$  și  $x + y < z - 1.0$ . Valorile curente ale variabilelor  $x$ ,  $y$  și  $z$  sînt citite de la tastatură.

```
Program P38;
{ Evaluarea expresiilor }
var x, y, z : real;
begin
 writeln('Introduceți numerele reale x, y, z:');
 readln(x, y, z);
 writeln(x * y + z);
 writeln(x + y < z - 1.0);
end.
```

### Întrebări și exerciții

❶ Fie  $x=1$ ,  $y=2$  și  $z=3$ . Evaluați următoarele expresii:

- |                        |                                                               |
|------------------------|---------------------------------------------------------------|
| a) $x + y + 2 * z$ ;   | f) $x * (y + y * z)$ ;                                        |
| b) $(x + y + 2) * z$ ; | g) $x * y < y * z$ ;                                          |
| c) $x * y + y * z$ ;   | h) $(x > y) \text{ or } (6 * x > y + z)$ ;                    |
| d) $x * (y + y) * z$ ; | i) <b>not</b> $(x + y + z > 0)$ ;                             |
| e) $(x * y + y) * z$ ; | j) <b>not</b> $(x + y > 0)$ <b>and</b> <b>not</b> $(z < 0)$ . |

❷ Care sînt regulile de evaluare a unei expresii PASCAL?

❸ Indicați prioritatea fiecărui operator al limbajului PASCAL.

❹ Precizați ordinea în care se calculează componentele unei expresii PASCAL.

❺ Elaborați un program care evaluează expresiile c și g din exercițiul 1. Valorile curente ale variabilelor reale  $x$ ,  $y$  și  $z$  se citesc de la tastatură.



### 3.4. TIPUL EXPRESIILOR PASCAL

În funcție de mulțimea valorilor pe care le poate lua, fiecare expresie se asociază cu un anumit tip de date. Conform conceptului de dată realizat în limbajul PASCAL, **tipul expresiei** derivă (rezultă) din tipul operanzilor și operatorilor care acționează asupra acestora. Prin urmare, tipul unei expresii poate fi dedus fără a calcula valoarea ei.

Tipul rezultatelor furnizate de operatori este indicat în *tabelul 3.3*. *Tabelul 3.4* conține tipul rezultatelor furnizate de funcțiile predefinite ale limbajului PASCAL.

*Tabelul 3.3. Tipul rezultatelor furnizate de operatori*

| OPERATOR            | TIPUL OPERANZILOR        | TIPUL REZULTATULUI |
|---------------------|--------------------------|--------------------|
| +, -, *             | integer integer          | integer            |
|                     | unul integer, altul real | real               |
| /                   | integer sau real         | real               |
| div                 | integer integer          | integer            |
| mod                 | integer integer          | integer            |
| not, and, or        | boolean boolean          | boolean            |
| <, <=, =, >=, >, <> | tipuri identice          | boolean            |
|                     | tipuri compatibile       | boolean            |
|                     | unul integer, altul real | boolean            |

Indiferent de tipul operanzilor, operatorul / (împărțirea) furnizează numai rezultate de tip `real`, iar operatorii relaționali — numai rezultate de tip `boolean`.

Pentru a afla tipul unei expresii, factorii, termenii și expresiile simple se examinează în ordinea evaluării lor. Tipul fiecărei părți componente se deduce cu ajutorul *tabelelor 3.3 și 3.4*.

De exemplu, fie expresia:

`(x>i) or (6*i<sin(x/y))`,

unde `i` este de tipul `integer`, iar `x` și `y` de tipul `real`.

Aflăm tipul fiecărei părți componente și al expresiei în ansamblu în ordinea de evaluare:

- |                                               |                        |
|-----------------------------------------------|------------------------|
| 1) <code>x&gt;i</code>                        | <code>boolean</code> ; |
| 2) <code>6*i</code>                           | <code>integer</code> ; |
| 3) <code>x/y</code>                           | <code>real</code> ;    |
| 4) <code>sin(x/y)</code>                      | <code>real</code> ;    |
| 5) <code>6*i&lt;sin(x/y)</code>               | <code>boolean</code> ; |
| 6) <code>(x&gt;i) or (6*i&lt;sin(x/y))</code> | <code>boolean</code> . |



Tabelul 3.4. Tipul rezultatelor furnizate de funcțiile predefinite

| <i>FUNCȚIA</i> | <i>TIPUL ARGUMENTULUI</i> | <i>TIPUL REZULTATULUI</i> |
|----------------|---------------------------|---------------------------|
| abs (x)        | integer sau real          | coincide cu tipul lui x   |
| sin (x)        | integer sau real          | real                      |
| cos (x)        | integer sau real          | real                      |
| arctan (x)     | integer sau real          | real                      |
| sqr (x)        | integer sau real          | coincide cu tipul lui x   |
| sqrt (x)       | integer sau real          | real                      |
| exp (x)        | integer sau real          | real                      |
| ln (x)         | integer sau real          | real                      |
| round (x)      | real                      | integer                   |
| trunc (x)      | real                      | integer                   |
| odd (i)        | integer                   | boolean                   |
| ord (v)        | ordinal                   | integer                   |
| pred (v)       | ordinal                   | coincide cu tipul lui v   |
| succ (v)       | ordinal                   | coincide cu tipul lui v   |
| chr (i)        | integer                   | char                      |
| eof (f)        | fișier                    | boolean                   |
| eoln (f)       | fișier                    | boolean                   |

Prin urmare, expresia în studiu este de tip boolean.

Întrucît în procesul deducției valorile concrete ale expresiilor în studiu nu se calculează, tipurile subdomeniu se extind la tipurile de bază.

De exemplu, în prezența declarațiilor

```
Type T1=1..10; { subdomeniu de integer}
 T2=11..20; { subdomeniu de integer}
var i:T1;
 j:T2;
```

avem:

| <u>expresie</u> | <u>tipul expresiei</u> |
|-----------------|------------------------|
| i+j             | integer                |
| i mod j         | integer                |
| i/j             | real                   |
| sin (i+j)       | real                   |
| i>j             | boolean                |
| ș.a.m.d.        |                        |

În funcție de tipul expresiei distingem:

- expresii aritmetice (integer sau real);
- expresii ordinale (integer, boolean, char, enumerare);
- expresii booleene (boolean).

De obicei, expresiile aritmetice se utilizează în calcule (instrucțiunea de atribuire), expresiile ordinale — în instrucțiunile **case** și **for**, iar expresiile booleene — în instrucțiunile **if**, **repeat** și **while**.

## Întrebări și exerciții

- ❶ Prin ce metodă se află tipul unei expresii PASCAL?
- ❷ În prezența declarațiilor:

```
var x, y : real;
 i, j : integer;
 p, q : boolean;
 r : char;
 s : (A, B, C, D, E, F, G, H);
```

aflați tipul următoarelor expresii:

- |                                              |                                          |
|----------------------------------------------|------------------------------------------|
| a) $i \bmod 3$ ;                             | i) $\text{sqr}(i) - \text{sqr}(j)$ ;     |
| b) $i/3$ ;                                   | j) $\text{sqr}(x) - \text{sqr}(y)$ ;     |
| c) $i \bmod 3 > j \text{ div } 4$ ;          | k) $\text{trunc}(x) + \text{trunc}(y)$ ; |
| d) $x + y / (x - y)$ ;                       | l) $\text{chr}(i)$ ;                     |
| e) <b>not</b> ( $x < i$ );                   | m) $\text{ord}(r)$ ;                     |
| f) $\sin(\text{abs}(i) + \text{abs}(j))$ ;   | o) $\text{ord}(s) > \text{ord}(r)$ ;     |
| g) $\sin(\text{abs}(x) + \text{abs}(y))$ ;   | p) $\text{pred}(E)$ ;                    |
| h) $p \text{ and } (\cos(x) \leq \sin(y))$ ; | q) $(-x + \sin(x - y)) / (2 * i)$ .      |

- ❸ Tipul unei expresii poate fi aflat din forma textuală a rezultatelor afișate pe ecran de instrucțiunea

`writeln(<Expresie> )`

*Exemple:*

| <u>rezultatul afișat pe ecran</u> | <u>tipul expresiei</u> |
|-----------------------------------|------------------------|
| 100                               | integer                |
| 1.000000000000E+02                | real                   |
| true                              | boolean                |

Elaborați programele respective și precizați tipul expresiilor ce urmează, pornind de la forma textuală a rezultatelor afișate:

- |                       |                                              |
|-----------------------|----------------------------------------------|
| a) $1 + 1.0$ ;        | f) <b>not</b> ( $x > y$ );                   |
| b) $1 / 1 + 1$ ;      | g) $\text{pred}(9) > \text{succ}(7)$ ;       |
| c) $9 * 3 \bmod 4$ ;  | h) $15 \text{ div } \text{ord}(3)$ ;         |
| d) $4 * x > 9 * y$ ;  | i) $\text{trunc}(x) + \text{round}(6 * y)$ ; |
| e) $\text{chr}(65)$ ; | j) $\text{sqr}(3) - \text{sqrt}(16)$ .       |

Se consideră că variabilele  $x$  și  $y$  sînt de tip real.

- ❹ Se consideră declarațiile

```
Type T1=1..10;
 T2=11..20;
 T3='A'..'Z';
 T4=(A, B, C, D, E, F, G, H);
```

```

var i:T1;
 j:T2;
 k:T3;
 m:'C'...'G';
 n:T4;
 p:C..G;
 q:boolean;

```

Aflați tipul următoarelor expresii:

- |                         |                                        |
|-------------------------|----------------------------------------|
| a) $i-j$ ;              | j) $\text{ord}(m)$ ;                   |
| b) $i \text{ div } j$ ; | k) $n>p$ ;                             |
| c) $6.3*i$ ;            | l) $\text{ord}(n)$ ;                   |
| d) $\cos(3*i-6*j)$ ;    | m) $\text{succ}(n)$ ;                  |
| e) $4*i>5*j$ ;          | n) $\text{pred}(p)$ ;                  |
| f) $k<m$ ;              | o) $\text{ord}(p)$ ;                   |
| g) $k<>m$ ;             | p) $\text{ord}(k)>\text{ord}(m)$ ;     |
| h) $\text{chr}(i)$ ;    | q) $(i>j) \text{ and } q$ ;            |
| i) $\text{ord}(k)$ ;    | r) $\text{not}(i+j>0) \text{ or } q$ . |

### 3.5. INSTRUCȚIUNEA DE ATRIBUIRE

Instrucțiunea în studiu are forma:

*<Variabilă> ::= <Expresie>*

Execuția unei instrucțiuni de atribuire presupune:

- evaluarea expresiei din partea dreaptă;
- atribuirea valorii obținute variabilei din partea stîngă.

*Exemple:*

```

x:=1
y:=x+3
z:=sin(x)+cos(y)
p:=not q
q:=(a<b) or (x<y)
c:='A'

```

De reținut că simbolul “:=” (se citește “atribuire”) desemnează o atribuire și nu trebuie confundat cu operatorul de relație “=” (egal).

O atribuire are loc dacă variabila și rezultatul evaluării expresiei sînt compatibile din punct de vedere al atribuirii. În caz contrar, se va declanșa o eroare.

Variabila și rezultatul evaluării expresiei sînt **compatibile din punct de vedere al atribuirii** dacă este adevărată una din următoarele afirmații:

- variabila și rezultatul evaluării sînt de tipuri identice;
- tipul rezultatului este un subdomeniu al tipului variabilei;



3) ambele tipuri sînt subdomenii ale aceluiași tip, iar rezultatul este în subdomeniul variabilei;

4) variabila este de tip real, iar rezultatul – de tip integer sau un subdomeniu al acestuia.

Pentru exemplificare, să considerăm următorul program:

```
Program P39;
{ Compatibilitate din punctul de vedere al atribuirii}
type T1=1..10; { subdomeniu de integer }
 T2=5..15; { subdomeniu de integer }
var i : T1;
 j : T2;
 k, m, n : integer;
 x : real;
begin
 write('k='); readln(k);
 i:=k; { corect pentru 1<=k<=10 }
 write('m='); readln(m);
 j:=m; { corect pentru 5<=m<=15 }
 write('n='); readln(n);
 i:=n+5; { corect pentru -4<=n<=5 }
 j:=n+2; { corect pentru 3<=n<=13 }
 x:=i+j;
 writeln('x=', x);
end.
```

Programul va derula fără erori numai pentru următoarele valori de intrare:

$$1 \leq k \leq 10;$$

$$5 \leq m \leq 15;$$

$$3 \leq n \leq 5.$$

Evident, în programul P39 instrucțiunile de tipul

```
k:=x;
i:=x+1;
j:=sin(i)
```

ș.a.m.d. ar fi incorecte întrucît  $x$ ,  $x+1$ ,  $\sin(i)$  sînt expresii de tip real, iar variabilele  $k$ ,  $i$ ,  $j$  sînt de tip integer sau subdomenii ale acestuia.

## Întrebări și exerciții

- ❶ Cum se execută o instrucțiune de atribuire?
- ❷ Explicați termenul “compatibilitate din punct de vedere al atribuirii”.
- ❸ Se consideră următoarele declarații:

```
Type Zi=(L, Ma, Mi, J, V, S, D);
 Culoare=(Galben, Verde, Albastru, Violet);
var i, j, k: integer;
```

```
z: Zi;
c: Culoare;
x: real;
```

Care din instrucțiunile ce urmează sînt corecte?

- |                 |                     |
|-----------------|---------------------|
| a) i:=12;       | f) c:=Verde;        |
| b) j:=ord(i);   | g) z:=D;            |
| c) x:=ord(z)+1; | h) c:=Pred(Galben); |
| d) k:=ord(x)+2; | i) x:=Succ(z);      |
| e) c:=i+4;      | j) i:=Succ(c).      |

- ④ Precizați pentru care valori ale variabilei j programul P40 va derula fără erori?

```
Program P40;
 var i : -10..+10; j : integer;
 begin
 write('j='); readln(j);
 i:=j+15;
 writeln('i=', i);
 end.
```

### 3.6. INSTRUCȚIUNEA APEL DE PROCEDURĂ

**Procedura** este un subalgoritm scris în limbaj de programare ce poate fi apelat din mai multe puncte ale unui program. Fiecare procedură are un nume, de exemplu, readln, writeln, CitireDate, A15 ș.a.m.d. Limbajul PASCAL include un set de proceduri predefinite, cunoscute oricărui program: read, readln, write, writeln, get, put, new etc. În completare, programatorul poate defini proceduri proprii.

Instrucțiunea *apel de procedură* lansează în execuție procedura cu numele specificat. Sintaxa instrucțiunii în studiu este

```
<Apel procedură> ::=
<Nume procedură> [<Listă parametri actuali>];
<Nume procedură> ::= <Identificator>
<Listă parametri actuali> ::=
 (<Parametru actual> {, <Parametru actual> })
```

Diagramele sintactice ale formulelor în studiu sînt prezentate în *fig. 3.4*.

În mod obișnuit, <Parametrul actual> este o expresie.

*Exemple:*

```
readln(x);
readln(x, y, z);
```

```

CitireDate(f, t);
Exit;
writeln(x+y, sin(x))

```

Tipul parametrilor actuali și ordinea în care aceștia apar în listă sînt impuse de declarațiile procedurii respective. În consecință, regulile de formare a listelor de parametri actuali vor fi studiate mai amănunțit în capitolele următoare.

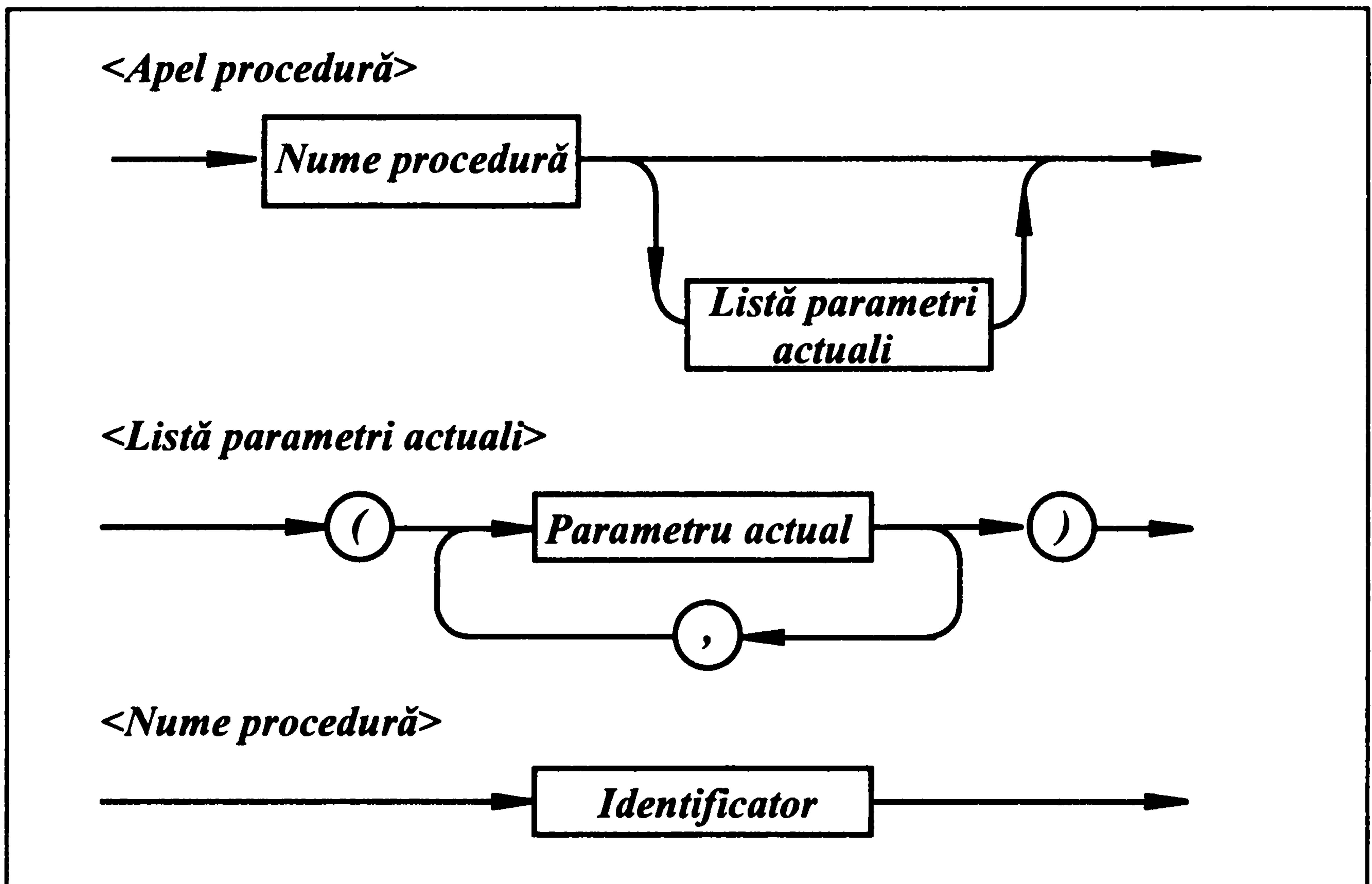


Fig. 3.4. Diagramele sintactice ale instrucțiunii apel de procedură

## Întrebări și exerciții

- ❶ Care este destinația instrucțiunii *apel de procedură*?
- ❷ Se consideră următoarele instrucțiuni:

```

readln(x, y, z, q);
CitireDate(ff, tt);
Halt;
writeln('x=', x, 'y=', y);
writeln('x+y=', x+y, 'sin(x)=' , sin(x))

```

Precizați numele procedurilor apelate, numărul de parametri actuali din fiecare apel și parametrii propriu-ziși.

- ❸ Indicați pe diagramele sintactice din fig. 3.4 drumurile ce corespund instrucțiunilor din exercițiul 2.



### 3.7. AFIȘAREA INFORMAȚIEI ALFANUMERICE

În versiunile uzuale ale limbajului PASCAL ecranul vizualizatorului este desemnat ca dispozitiv-standard de ieșire. De regulă, ecranul este împărțit în zone convenționale, numite zone-caracter. De obicei, aceste zone formează 25 de linii — câte 80 de caractere pe linie. Zona în care va fi afișat caracterul curent este indicată de cursor.

Datele de ieșire ale unui program PASCAL pot fi afișate pe ecran printr-un apel

`write(x)` sau `writeln(x)`.

Apelul

`write(x1, x2, ..., xn)`.

este echivalent cu `write(x1); write(x2); ...; write(xn)`

Parametrii actuali dintr-un apel `write` sau `writeln` se numesc **parametri de ieșire**. Aceștia pot avea una din formele:

$e$

$e:w$

$e:w:f$

unde  $e$  este o expresie de tip `integer`, `real`, `boolean`, `char` sau șir de caractere a cărei valoare trebuie afișată;  $w$  și  $f$  sînt expresii de tip `integer`, numite **specificatori de format**.

Expresia  $w$  specifică prin valoarea sa numărul minim de caractere ce vor fi folosite la afișarea valorii lui  $e$ ; dacă sînt necesare mai puțin de  $w$  caractere, atunci forma externă a valorii lui  $e$  va fi completată cu spații la stînga pînă la  $w$  caractere (*fig. 3.5*).

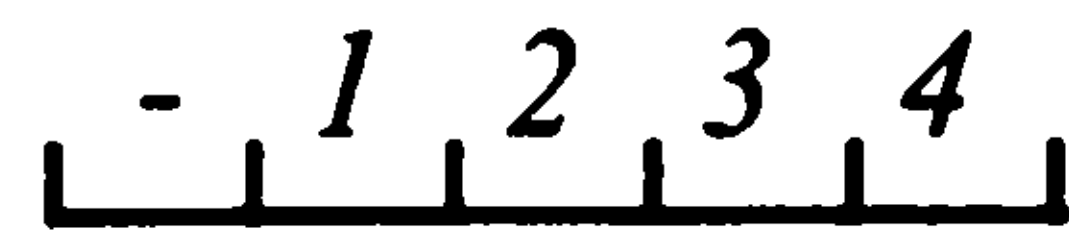
Specificatorul de format  $f$  are sens în cazul în care  $e$  este de tip `real` și indică numărul de cifre care urmează punctul zecimal în scrierea valorii lui  $e$  în virgulă fixă, fără factor de scală. În lipsa lui  $f$  valoarea lui  $e$  se scrie în virgulă mobilă, cu factor de scală (*fig. 3.6*).

Diferența dintre procedurile `write` și `writeln` constă în faptul că, după afișarea datelor, `write` lasă cursorul în linia curentă, în timp ce `writeln` îl trece la începutul unei linii noi. Utilizarea rațională a apelurilor `write`, `writeln` și a specificatorilor de format asigură o afișare lizibilă a datelor de ieșire. Dacă pe ecran se afișează mai multe valori, se recomandă ca acestea să fie însoțite de identificatorii respectivi sau de mesaje sugestive.

*Exemple:*

```
write('Suma numerelor introduse este');
writeln(s:20);
writeln('Suma=', s);
writeln('s=' s);
writeln('x=', x, 'y=':5, y, 'z=':5, z)
```

*write (-1234)*



*write (-1234:10)*



*write ('a')*



*write ('a':5)*



*write ('a', 'b')*



*write ('a':5, 'b':5)*

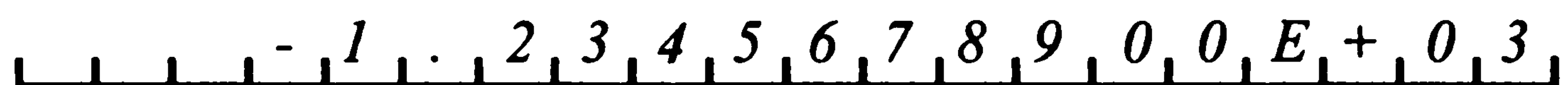


*Fig. 3.5. Semnificația specificatorului de format w*

*write (-1234.567890)*



*write (-1234.567890:20)*



*write (-1234.567890:20:1)*



*write (-1234.567890:20:4)*



*Fig. 3.6. Semnificația specificatorului de format f*

## Întrebări și exerciții

- ❶ Care este destinația specificatorului de format?

- ② Cum se numesc parametrii actuali ai unui apel de procedură `write` sau `writeln`?
- ③ Precizați formatul datelor afișate pe ecran de programele ce urmează:

```
Program P41;
{ Afișarea datelor de tip integer }
var i : integer;
begin
 i:=-1234;
 writeln(i);
 writeln(i:1);
 writeln(i:8);
 writeln(i, i);
 writeln(i:8, i:8);
 writeln(i, i, i);
 writeln(i:8, i:8, i:8);
end.
```

```
Program P42;
{ Afișarea datelor de tip real }
var x : real;
begin
 x:=-1234.567890;
 writeln(x);
 writeln(x:20);
 writeln(x:20:1);
 writeln(x:20:2);
 writeln(x:20:4);
 writeln(x, x, x);
 writeln(x:20, x:20, x:20);
 writeln(x:20:4, x:20:4, x:20:4);
end.
```

```
Program P43;
{ Afișarea datelor de tip boolean }
var p : boolean;
begin
 p:=false;
 writeln(p);
 writeln(p:10);
 writeln(p, p);
 writeln(p:10, p:10);
end.
```

```
Program P44;
{ Afișarea șirurilor de caractere }
begin
 writeln('abc');
 writeln('abc':10);
 writeln('abc', 'abc');
```



```
writeln('abc':10, 'abc':10);
end.
```

- ④ Elaborați un program care afișează pe ecran valorile 1234567890, 123, 123.0 și true după cum urmează:

```
1234567890
123
123.0
true
1234567890
123
123.000
true
```

### 3.8. CITIREA DATELOR DE LA TASTATURĂ

În mod obișnuit, tastatura vizualizatorului este desemnată ca **dispozitiv-standard de intrare**. Citirea datelor de la tastatură se realizează prin apelul procedurilor predefinite `read` sau `readln`. Lista parametrilor actuali a unui apel `read` sau `readln` poate să includă variabile de tip `integer`, `real`, `char` și șir de caractere.

Apelul

```
read(x)
```

are următorul efect. Dacă variabila `x` este de tip `integer` sau `real`, atunci este citit întregul șir de caractere care reprezintă valoarea întreagă sau reală. Dacă `x` este de tip `char`, procedura citește un singur caracter.

Apelul

```
read(x1, x2, ..., xn)
```

este echivalent cu

```
read(x1); read(x2); ...; read(xn).
```

Datele numerice introduse de la tastatură trebuie separate prin spații sau caractere sfârșit de linie. Spațiile dinaintea unei valori numerice sînt ignorate. Șirul de caractere care reprezintă o valoare numerică se conformează sintaxei constantelor numerice de tipul respectiv. În caz contrar, este semnalată o eroare de intrare-ieșire.

De exemplu, fie programul:

```
Program P45;
{ Citirea datelor numerice de la tastatură }
var i, j : integer;
x, y : real;
```

```

begin
 read(i, j, x, y);
 writeln('Ați introdus:');
 writeln('i=', i);
 writeln('j=', j);
 writeln('x=', x);
 writeln('y=', y);
end.

```

în care sînt citite de la tastatură valorile variabilelor *i*, *j*, *x*, *y*. După lansarea programului în execuție, utilizatorul tastează:

```

1<ENTER>
2<ENTER>
3.0<ENTER>
4.0<ENTER>

```

Pe ecran se va afișa:

```

Ați introdus:
i=1
j=2
x=3.000000000000E+00
y=4.000000000000E+00

```

Același efect se va obține și la tastarea numerelor într-o singură linie:

```

1 2 3.0 4.0<ENTER>

```

Dacă e necesar, numerele întregi, introduse de utilizator, sînt convertite în valori reale. De exemplu, în cazul programului P45 utilizatorul poate tasta

```

1 2 3 4<ENTER>

```

Procedura `readln` citește datele în același mod ca și procedura `read`. Însă, după citirea ultimei valori, restul caracterelor din linia curentă se ignoră. Pentru exemplificare, prezentăm programul P46:

```

Program P46;
{ Apelul procedurii readln }
var i,j : integer;
 x,y : real;
begin
 writeln('Apelul procedurii read');
 read(i, j);
 read(x, y);
 writeln('Ați introdus:');
 writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
 writeln('Apelul procedurii readln');

```

```
readln(i, j);
readln(x, y);
writeln('Ați introdus:');
writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
end.
```

La execuția instrucțiunilor

```
read(i, j);
read(x, y)
```

valorile numerice din linia introdusă de utilizator

1 2 3 4<ENTER>

vor fi atribuite variabilelor respectiv i, j, x, y. La execuția instrucțiunii

```
readln(i, j)
```

valorile numerice 1 și 2 din linia

1 2 3 4<ENTER>

vor fi atribuite variabilelor i și j. Numerele 3 și 4 se ignoră. În continuare calculatorul execută instrucțiunea

```
readln(x, y)
```

adică va aștepta introducerea unor valori pentru x și y.

Subliniem faptul că apelul procedurii readln fără parametri va forța calculatorul să aștepte acționarea tastei <ENTER>. Acest apel se utilizează pentru a suspenda derularea programului, oferindu-i utilizatorului posibilitatea să analizeze rezultatele afișate anterior pe ecran.

Pentru a înlesni introducerea datelor, se recomandă ca apelurile read(...) și readln(...) să fie precedate de afișarea unor mesaje sugestive.

*Exemple:*

```
write('Dați două numere:'); readln(x, y);
write('Dați un număr întreg:'); readln(i);
write('x='); readln(x);
write('Răspundeți D sau N:'); readln(c);
```

## Întrebări și exerciții

- ❶ Cum se separă datele numerice ce se introduc de la tastatură?
- ❷ Care este diferența dintre procedurile read și readln?
- ❸ Se consideră următorul program:

```
Program P47;
var i : integer;
 c : char;
 x : real;
begin
```



```

readln(i);
readln(c);
readln(x);
writeln('i=', i);
writeln('c=', c);
writeln('x=', x);
readln;
end.

```

Precizați rezultatele afișate de acest program după tastarea următoarelor date de intrare:

|      |          |        |                |
|------|----------|--------|----------------|
| a) 1 | b) 1 2 3 | c) 123 | d) 123 456 789 |
| 2    | 5 6 7    | 456    | abc def ghi    |
| 3    | 8 9 0    | 789    | 890 abc def    |

### 3.9. INSTRUCȚIUNEA DE EFECT NUL

Executarea acestei instrucțiuni nu are nici un efect asupra variabilelor programului. Sintaxa instrucțiunii în studiu este:

*<Instrucțiunea de efect nul> ::=*

Prin urmare, în textul unui program instrucțiunea de efect nul nu este reprezentată prin nimic. Întrucât instrucțiunile unui program sînt despărțite între ele prin delimitatorul “;”, prezența instrucțiunii de efect nul este marcată de apariția acestui delimitator.

De exemplu, în textul

```
x:=4;;; y:=x+1
```

există 5 instrucțiuni dintre care 3 de efect nul.

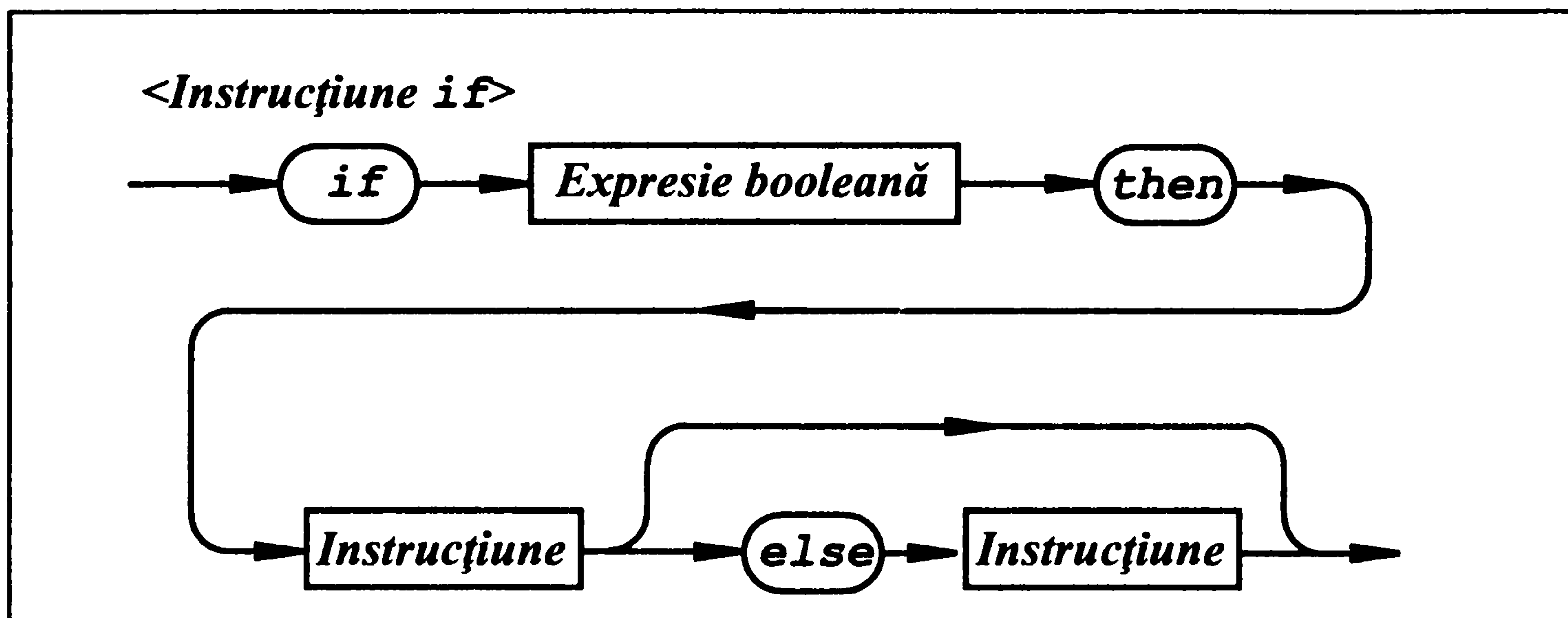
În mod obișnuit, instrucțiunea de efect nul se utilizează la etapa elaborării și depănării unor programe complexe. Deși efectul său la execuție este nul, inserarea sau eliminarea unei astfel de instrucțiuni (mai exact, a simbolului “;”) poate să altereze semnificația programului.

### 3.10. INSTRUCȚIUNEA IF

Instrucțiunea de ramificare simplă **if**, în funcție de valoarea unei expresii de tip boolean, decide fluxul execuției. Sintaxa instrucțiunii este:

*<Instrucțiune if> ::=*  
**if** *<Expresie booleană>* **then** *<Instrucțiune>*  
**[else** *<Instrucțiune>* **]**

Diagrama sintactică a instrucțiunii în studiu este prezentată în *fig. 3.7*. Expresia booleană din componența instrucțiunii **if** se numește **condiție**.



*Fig. 3.7.* Diagrama sintactică <Instrucțiune **if**>

Execuția instrucțiunii **if** începe prin evaluarea condiției. Dacă rezultatul evaluării este **true**, atunci se execută instrucțiunea situată după cuvântul-cheie **then**. Dacă condiția are valoarea **false**, atunci: sau se execută instrucțiunea situată după cuvântul-cheie **else** (dacă există), sau se trece la instrucțiunea situată după instrucțiunea **if**.

În programul ce urmează instrucțiunea **if** este utilizată pentru determinarea maximului a două numere *x* și *y*, citite de la tastatură.

```

Program P48;
{ Determinarea maximului a două numere }
var x, y, max : real;
begin
 writeln('Dați două numere:');
 write('x='); readln(x);
 write('y='); readln(y);
 if x>=y then max:=x else max:=y;
 writeln('max=', max);
 readln;
end.

```

Următorul program transformă cifrele romane *I* (unu), *V* (cinci), *X* (zece), *L* (cincizeci), *C* (o sută), *D* (cinci sute) sau *M* (o mie), citite de la tastatură, în numerele corespunzătoare din sistemul zecimal.

```

Program P49;
{ Conversia cifrelor romane }
var i : integer; c : char;
begin
 i:=0;

```

```

writeln('Introduceți una din cifrele');
writeln('romane I, V, X, L, C, D, M');
readln(c);
if c='I' then i:=1;
if c='V' then i:=5;
if c='X' then i:=10;
if c='L' then i:=50;
if c='C' then i:=100;
if c='D' then i:=500;
if c='M' then i:=1000;
if i=0 then
writeln(c, ' - nu este o cifră romană')
else writeln(i);
readln;
end.

```

De reținut că limbajul PASCAL nu consideră simbolul “;” ca făcând parte din instrucțiune, ci îl folosește ca delimitator. Prin urmare, dacă într-o instrucțiune

**if B then S**

introducem înainte de S instrucțiunea cu efect nul

**if B then; S**

atunci S nu mai intră în componența instrucțiunii condiționale, deci este executată indiferent de valoarea lui B.

Dacă într-o instrucțiune

**if B then I else J**

introducem după I simbolul “;”, obținem un program incorect în aspect sintactic:

**if B then I; else J**

În acest caz secvența **else J** este interpretată ca fiind instrucțiunea ce urmează celei condiționale.

## Întrebări și exerciții

- ❶ Care este destinația instrucțiunii **if**?
- ❷ Ce valori va lua variabila *x* după executarea fiecăreia dintre instrucțiunile ce urmează? Se consideră că *a*=18, *b*= -15 și *p*=true.
  - a) **if a>b then x:=1 else x:=4;**
  - b) **if a<b then x:=15 else x:=-21;**
  - c) **if p then x:=32 else x:=638;**
  - d) **if not p then x:=0 else x:=1;**
  - e) **if (a<b) and p then x:=-1 else x:=1;**
  - f) **if (a>b) or p then x:=-6 else x:=-5;**



- g) **if** not (a>b) **then** x:=19 **else** x:=-2;  
 h) **if** (a=b) or p **then** x:=89 **else** x:=-15.

③ Elaborați un program care calculează valorile uneia dintre funcțiile ce urmează:

$$a) y = \begin{cases} 2x, & x \geq 0; \\ \frac{x}{2}, & x < 0; \end{cases}$$

$$b) z = \begin{cases} \sin(x+y), & x > 2y; \\ \cos(x-y), & x \leq 2y; \end{cases}$$

$$c) y = \begin{cases} x, & x \geq 3; \\ x^2, & x < 3; \end{cases}$$

$$d) y = \begin{cases} x^2, & |x| > 5; \\ x^3, & |x| \leq 5; \end{cases}$$

*Exemplu:*  $y = \begin{cases} 2x^2 + 6, & x > 4; \\ x^3 - 3, & x \leq 4; \end{cases}$

```
Program P50;
var x,y : real;
begin
 write('x='); readln(x);
 if x>4 then y:=2*sqr(x)+6
 else y:=x*x*x-3;
 writeln('y=', y);
 readln;
end.
```

④ Ce rezultate va afișa următorul program?

```
Program P51;
var x,y : real;
begin
 write('x='); readln(x);
 y:=x;
 if x>0 then; y:=2*x;
 writeln('y=', y);
 readln;
end.
```

⑤ Comentați mesajele afișate pe ecran pe parcursul compilării programului P52:

```
Program P52;
{ Eroare }
var x,y : real;
begin
 write('x='); readln(x);
```

```

if x>4 then y:=2*sqr(x)+6;
else y:=x*x*x-3;
writeln('y=', y); readln;
end.

```

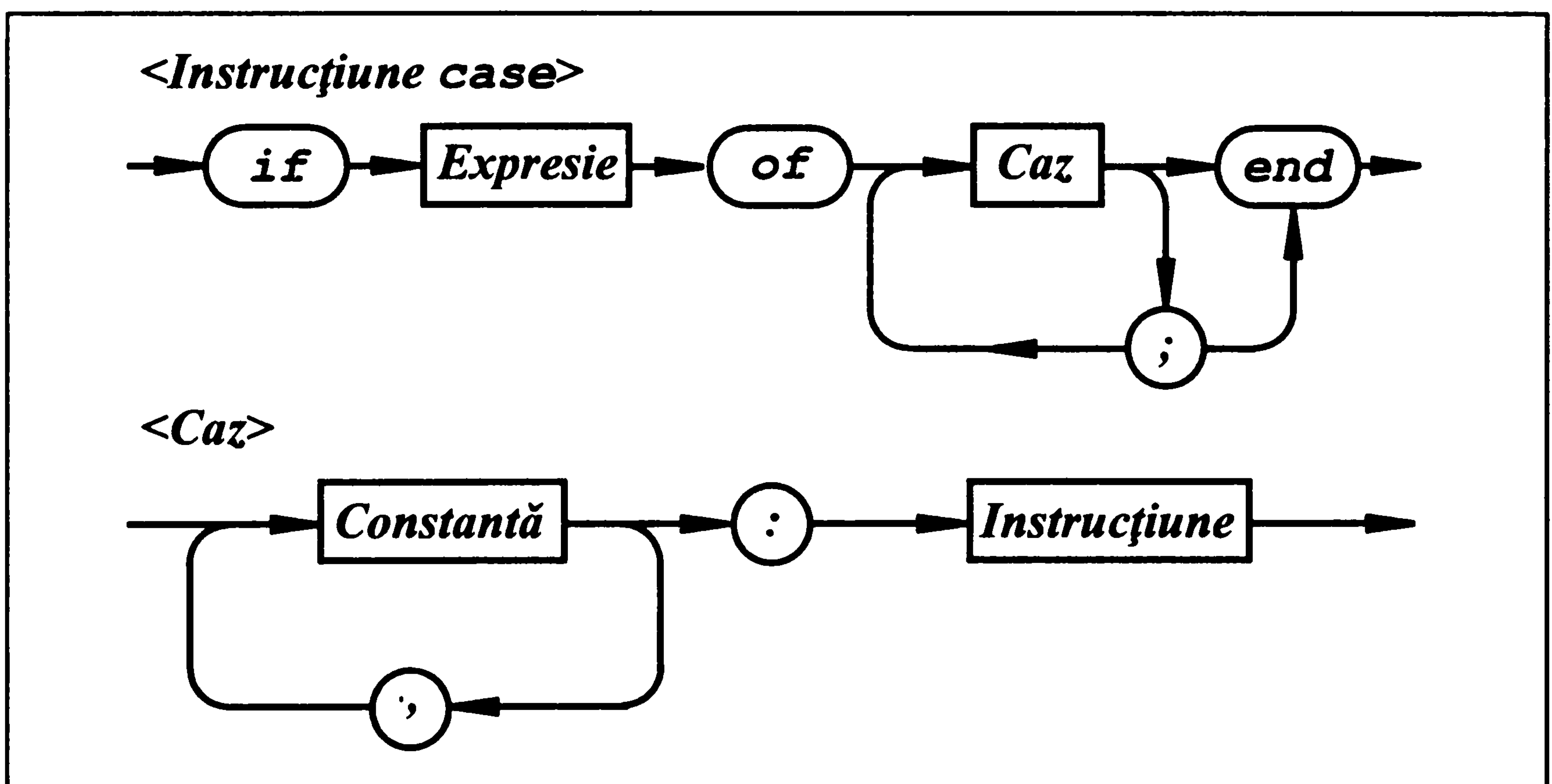
- ⑥ Scrieți un program care transformă numerele zecimale 1, 5, 10, 50, 100, 500 și 1000, citite de la tastatură, în cifre romane.

### 3.11. INSTRUCȚIUNEA CASE

Instrucțiunea de ramificare multiplă **case** conține o expresie numită **selector** și o listă de instrucțiuni. Fiecare instrucțiune este prefixată de una sau mai multe constante de caz. Sintaxa instrucțiunii în studiu este:

*<Instrucțiune case>* ::=  
**case** *<expresie>* **of** [*<Caz>* { ; *<Caz>* } ] [ ; ] **end**  
*<Caz>* ::= *<Constantă>* { , *<Constantă>* } : *<Instrucțiune>*

Diagramele sintactice corespunzătoare sînt prezentate în *fig. 3.8*.



*Fig. 3.8.* Diagramele sintactice ale instrucțiunii **case**

Selectorul trebuie să fie de tip ordinal. Toate constantele de caz trebuie să fie unice și compatibile din punct de vedere al atribuirii cu tipul selectorului.

*Exemple:*

```

var i : integer; c : char; a, b, y : real;
case i of
 0, 2, 4, 6, 8 : writeln('Cifră pară');
 1, 3, 5, 7, 9 : writeln('Cifră impară');

```

```

end;
case c of
 '+' : y:=a+b;
 '-' : y:=a-b;
 '*' : y:=a*b;
 '/' : y:=a/b;
end;

```

Execuția instrucțiunii **case** începe prin evaluarea selectorului. În funcție de valoarea obținută, se execută instrucțiunea prefixată de constanta respectivă.

În programul ce urmează instrucțiunea **case** este utilizată pentru conversia cifrelor romane în numere zecimale.

```

Program P53;
{ Conversia cifrelor romane }
var i : integer; c : char;
begin
 i:=0;
 writeln('Introduceți una din cifrele');
 writeln('romane I, V, X, L, C, D, M');
 readln(c);
 case c of
 'I' : i:=1;
 'V' : i:=5;
 'X' : i:=10;
 'L' : i:=50;
 'C' : i:=100;
 'D' : i:=500;
 'M' : i:=1000;
 end;
 if i=0 then
 writeln(c, ' - nu este o cifră romană')
 else writeln(i);
 readln;
end.

```

Subliniem faptul că în unele implementări ale limbajului sintaxa și semantica instrucțiunii **case** au fost modificate. Lista cazurilor poate să includă o instrucțiune precedată de cuvântul-cheie **else** (în unele versiuni **otherwise**). Constantele de caz pot fi înlocuite cu intervale de forma:

<Constantă> .. <Constantă>

*Exemplu* (Turbo Pascal 7.0):

```

Program P54;
{ Simularea unui calculator de buzunar }
var a,b : real; c : char;
begin
 write('a='); readln(a);

```



```

write('b='); readln(b);
write('Cod operație '); readln(c);
case c of
 '+' : writeln('a+b=', a+b);
 '-' : writeln('a-b=', a-b);
 '*' : writeln('a*b=', a*b);
 '/' : writeln('a/b=', a/b);
else writeln('Cod operație necunoscut');
end;
readln;
end.

```

## Întrebări și exerciții

- ❶ Indicați pe diagramele sintactice din *fig. 3.8* drumurile care corespund instrucțiunilor **case** din programele P53 și P54.
- ❷ Cum se execută o instrucțiune **case**? De ce tip trebuie să fie selectorul?
- ❸ Ce fel de constante pot fi utilizate ca constante de caz?
- ❹ Înlocuiți instrucțiunea **case** a programului P54 cu o secvență echivalentă de instrucțiuni **if**.
- ❺ Utilizând instrucțiunea **case**, scrieți un program care transformă numerele zecimale 1, 5, 10, 50, 100, 500, 1000, citite de la tastatură, în cifre romane.
- ❻ Ce va apare pe ecran în urma execuției programului P55?

```

Program P55;
type Semnal=(Rosu, Galben, Verde);
var s : Semnal;
begin
 s:=Verde;
 s:=pred(s);
 case s of
 Rosu : writeln('STOP');
 Galben : writeln('ATENȚIE');
 Verde : writeln('START');
 end;
 readln;
end.

```

- ❼ Comentați programele:

```

Program P56;
{ Eroare }
var x : real;
begin
 writeln('x='); readln(x);
 case x of
 0,2,4,6,8 : writeln('Cifră pară');
 1,3,5,7,9 : writeln('Cifră impară');
 end;
 readln;
end.

```

```

Program P57;
{ Eroare }
var i : 1..4;
begin
 write('i='); readln(i);
 case i of
 1 : writeln('unu');
 2 : writeln('doi');
 3 : writeln('trei');
 4 : writeln('patru');
 5 : writeln('cinci');
 end;
 readln;
end.

```

```

Program P58;
{ Eroare }
type Semnal=(Rosu, Galben, Verde);
 Culoare=(Albastru, Portocaliu);
var s : Semnal;
 c : Culoare;
begin
 { ... }
 case s of
 Rosu : writeln('STOP');
 Galben : Writeln('ATENȚIE');
 Verde : writeln('START');
 Albastru : writeln('PAUZĂ');
 end;
 { ... }
end.

```

### 3.12. INSTRUCȚIUNEA FOR

Instrucțiunea **for** indică execuția repetată a unei instrucțiuni în funcție de valoarea unei variabile de control. Sintaxa instrucțiunii în studiu este

*<Instrucțiune for> ::=*  
     **for** *<Variabilă>* ::= *<Expresie>* *<Pas>* *<Expresie>*  
     **do** *<Instrucțiune>*

*<Pas>* ::= **to** | **downto**

Diagramele sintactice sînt prezentate în *fig. 3.9*.

Variabila situată după cuvîntul-cheie **for** se numește **variabilă de control** sau **contor**. Această variabilă trebuie să fie de tip ordinal.

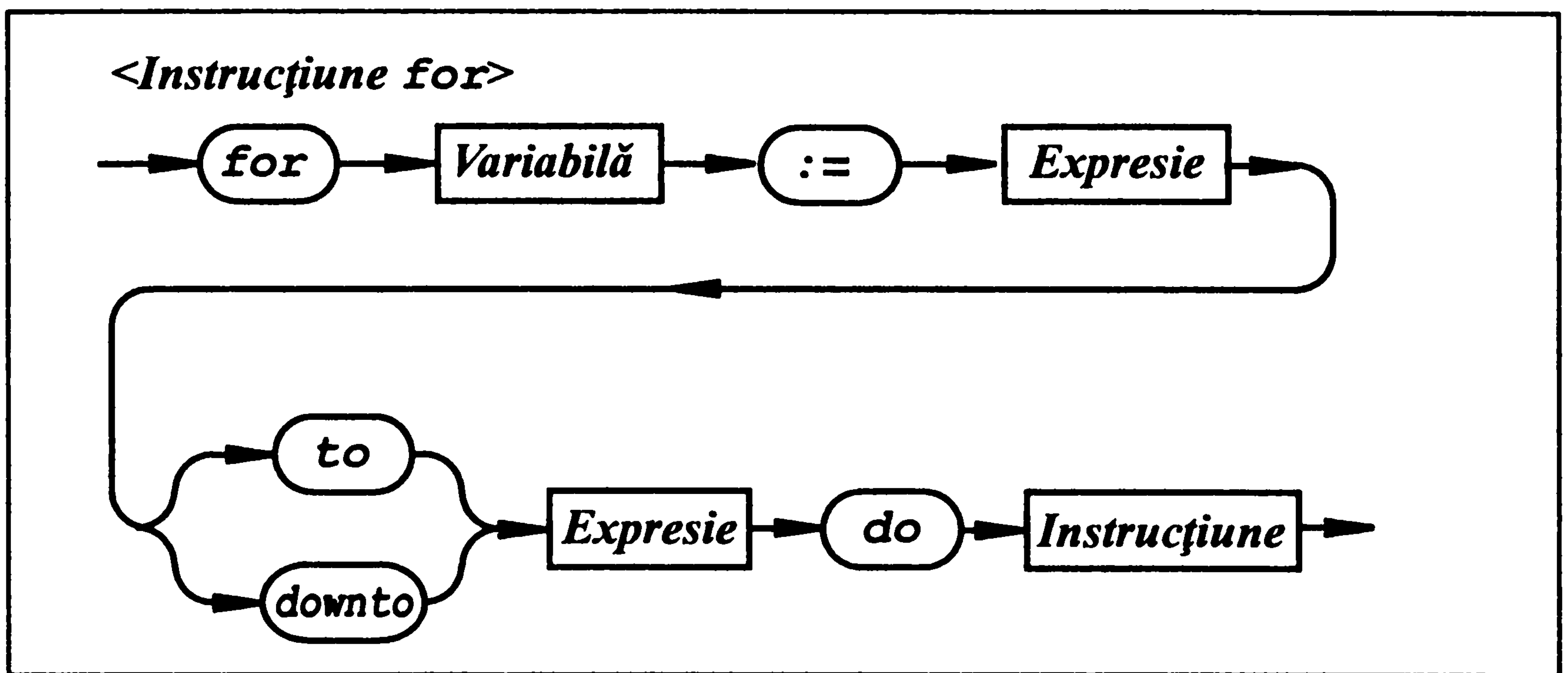


Fig. 3.9. Diagrama sintactică a instrucțiunii **for**

Valorile expresiilor din componența instrucțiunii **for** trebuie să fie compatibile, în aspectul atribuirii, cu tipul variabilei de control. Aceste expresii sînt evaluate o singură dată, la începutul ciclului. Prima expresie indică valoarea inițială, iar expresia a doua – valoarea finală a variabilei de control.

Instrucțiunea situată după cuvîntul-cheie **do** se execută pentru fiecare valoare din domeniul determinat de valoarea inițială și de valoarea finală.

Dacă instrucțiunea **for** utilizează pasul **to**, valorile variabilei de control sînt incrementate la fiecare repetiție, adică se trece la succesorul valorii curente. Dacă valoarea inițială este mai mare decît valoarea finală, instrucțiunea situată după cuvîntul-cheie **do** nu se execută nici o dată.

Dacă instrucțiunea **for** utilizează pasul **downto**, valorile variabilei de control sînt decrementate la fiecare repetiție, adică se trece la predecesorul valorii curente. Dacă valoarea inițială este mai mică decît valoarea finală, instrucțiunea situată după cuvîntul-cheie **do** nu se execută nici o dată.

*Exemplu :*

```

Program P59;
{ Instrucțiunea for }
var i : integer; c : char;
begin
 for i:=0 to 9 do write(i:2);
 writeln;
 for i:=9 downto 0 do write(i:2);
 writeln;
 for c:='A' to 'Z' do write(c:2);
 writeln;
 for c:='Z' downto 'A' do write(c:2);
 writeln;

```



```
 readln;
end.
```

Rezultatele afișate pe ecran:

```
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

Valorile variabilei de control nu pot fi modificate în interiorul ciclului, adică :

- 1) nu se fac atribuiri variabilei de control;
- 2) variabila actuală de control nu poate fi variabilă de control a altei instrucțiuni **for** incluse;
- 3) nu se admit apeluri de tipul `read`, `readln` în care apare variabila de control.

La ieșirea din instrucțiunea **for** valoarea variabilei de control nu este definită, în afara cazului când ieșirea din ciclu se face forțat, printr-o instrucțiune de salt necondiționat **goto**.

Instrucțiunea **for** este utilă pentru programarea algoritmilor iterativi, în care numărul de repetări este cunoscut. Pentru exemplificare prezentăm programele P60, P61 și P62 care calculează respectiv  $n!$ ,  $x^n$  și suma  $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots + \frac{1}{n}$ .

```
Program P60;
{ Calcularea factorialului }
var n, i, f : 0..MaxInt;
begin
 write('n='); readln(n);
 f:=1;
 for i:=1 to n do f:=f*i;
 writeln('n!=', f);
 readln;
end.
```

```
Program P61;
{ Calcularea lui x la puterea n }
var x, y : real;
 n, i : 0..MaxInt;
begin
 write('x='); readln(x);
 write('n='); readln(n);
 y:=1;
 for i:=1 to n do y:=y*x;
 writeln('y=', y);
 readln;
end.
```

```

Program P62;
{ Calcularea sumei $1 + 1/2 + 1/3 + \dots + 1/n$ }
var n, i : 1..MaxInt;
 s : real;
begin
 write('n=');
 readln(n);
 s:=0;
 for i:=1 to n do s:=s+1/i;
 writeln('s=', s);
 readln;
end.

```

## Întrebări și exerciții

- ❶ Indicați pe diagrama sintactică din *fig. 3.9* drumurile care corespund instrucțiunilor **for** din programul P59.
- ❷ Cum se execută o instrucțiune **for**?
- ❸ Ce va afișa pe ecran programul P63?

```

Program P63;
type Zi=(L, Ma, Mi, J, V, S, D);
var z : Zi;
begin
 for z:=L to S do writeln(ord(z));
 readln;
 for z:=D downto Ma do writeln(ord(z));
 readln;
end.

```

- ❹ Se consideră declarațiile :

```

var i, j, n : integer;
 x, y : real;
 c : char;

```

Care din instrucțiunile ce urmează sînt corecte?

- a) **for** i:=-5 **to** 5 **do** j:=i+3;
- b) **for** i:=-5 **to** 5 **do** i:=j+3;
- c) **for** j:=-5 **to** 5 **do** i:=j+3;
- d) **for** i:=1 **to** n **do** y:=y/i;
- e) **for** x:=1 **to** n **do** y:=y/x;
- f) **for** c:='A' **to** 'Z' **do** writeln(ord(c));
- g) **for** c:='Z' **downto** 'A' **do** writeln(ord(c));
- h) **for** i:=-5 **downto** -10 **do** readln(i);
- i) **for** i:=ord('A') **to** ord('A')+9 **do** writeln(i);
- j) **for** c:='0' **to** '9' **do** writeln(c, ord(c):3);
- k) **for** j:=i/2 **to** i/2+10 **do** writeln(j);

- ❺ Se consideră declarațiile:

**var** i, m, n : integer;

De câte ori se vor executa apelurile `writeln(i)` și `writeln(2*i)` din componența instrucțiunilor

**for** i:=m **to** n **do** `writeln(i)`;  
**for** i:=m **to** n **do** `writeln(2*i)`;

dacă:

- a) m=1, n=5;
- b) m=3, n=5;
- c) m=3, n=3;
- d) m=5, n=3?

⑥ Elaborați un program care afișează pe ecran codurile caracterelor 'A', 'B', 'C', ..., 'Z'.

⑦ Calculați pentru primii  $n$  termeni:

- a)  $1 + 3 + 5 + 7 + \dots$  și  $1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots$ ;
- b)  $2 + 4 + 6 + 8 + \dots$  și  $2 \cdot 4 \cdot 6 \cdot 8 \cdot \dots$ ;
- c)  $3 + 6 + 9 + 12 + \dots$  și  $3 \cdot 6 \cdot 9 \cdot 12 \cdot \dots$ ;
- d)  $4 + 8 + 12 + 16 + \dots$  și  $4 \cdot 8 \cdot 12 \cdot 16 \cdot \dots$ ;

*Exemplu:* Pentru  $n=3$  avem  $1 + 3 + 5 = 9$ ;  $1 \cdot 3 \cdot 5 = 15$ .

⑧ Calculați suma primilor  $n$  termeni:

$$1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} \dots$$

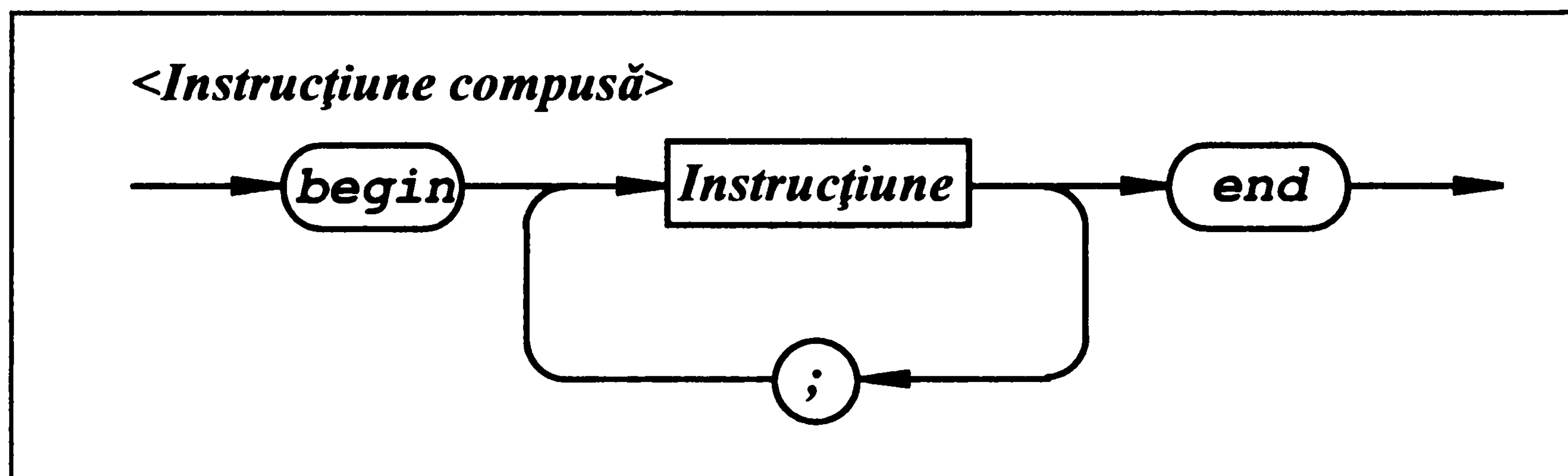
*Indicație:* Utilizați în ciclu instrucțiunea `if odd(...) then ... else ...`.

### 3.13. INSTRUCȚIUNEA COMPUSĂ

Instrucțiunea în studiu are următoarea sintaxă:

*<Instrucțiune compusă>* ::=  
**begin** *<Instrucțiune>* { ; *<Instrucțiune>* } **end**

Diagrama sintactică este prezentată în *fig. 3.10*.



*Fig. 3.10.* Diagrama sintactică *<Instrucțiune compusă>*



*Exemple:*

```
1) begin
 a:=x+12;
 p:=q and r;
 writeln(p)
 end;
2) begin
 write('x=');
 readln(x)
 end;
```

Cuvintele-cheie **begin** și **end** joacă rolul unor “paranteze”. Mulțimea instrucțiunilor cuprinse între aceste paranteze, din punctul de vedere al limbajului, formează o singură instrucțiune. Prin urmare, instrucțiunea compusă este utilă, pentru a plasa mai multe instrucțiuni în locurile din programe în care este permisă numai o singură instrucțiune (vezi instrucțiunile **if**, **case**, **for** ș.a.).

*Exemple:*

```
1) if a>0 then begin
 x:=a+b;
 y:=a*b
 end;
 else begin
 x:=a-b;
 y:=a/b
 end;
2) case c of
 '+' : begin y:=a+b; writeln('Adunarea') end;
 '-' : begin y:=a-b; writeln('Scăderea') end;
 '*' : begin y:=a*b; writeln('Înmulțirea') end;
 '/' : begin y:=a/b; writeln('Împărțirea') end;
 end ;
3) for i:=1 to n do
 begin
 write('x=');
 readln(x);
 s:=s+x
 end;
```

De menționat că partea executabilă a oricărui program este o instrucțiune compusă, adică o secvență de instrucțiuni încadrată între “parantezele” **begin** și **end**.

Întrucât simbolul “;” nu termină, ci separă instrucțiunile, el nu este necesar înaintea cuvîntului-cheie **end**. Cu toate acestea, mulți programatori inserează acest simbol cu scopul de a continua, în caz de necesitate, lista respectivă de instrucțiuni. Amintim că apariția adițională a unui simbol “;” semnifică inserarea unei instrucțiuni de efect nul.

Pentru a face programele mai lizibile, cuvintele **begin** și **end** se scriu strict unul sub altul, iar instrucțiunile dintre “paranteze” — cu câteva poziții mai la dreapta. Dacă instrucțiunea compusă **begin...end** este inclusă în componența altor instrucțiuni (**if**, **case**, **for** ș.a.), cuvintele-cheie **begin** și **end** se scriu deplasate la dreapta.

Pentru exemplificare prezentăm programul P64 în care se calculează media aritmetică a  $n$  numere, citite de la tastatură.

```
Program P64;
{Media aritmetică a n numere }
var x, suma, media : real; i, n : integer;
begin
 write('n='); readln(n);
 suma:=0;
 writeln('Dați ', n, ' numere:');
 for i:=1 to n do
 begin
 write('x='); readln(x);
 suma:=suma+x;
 end;
 if n>0 then
 begin
 media:=suma/n;
 writeln('media=', media);
 end
 else writeln('media=*****');
 readln;
end.
```

## Întrebări și exerciții

- ❶ Care este destinația instrucțiunii compuse?
- ❷ Indicați pe diagrama sintactică din *fig. 3.10* drumurile care corespund instrucțiunii compuse din programul P64.
- ❸ Elaborați un program care citește de la tastatură  $n$  numere și afișează pe ecran:
  - a) suma și media aritmetică a numerelor citite;
  - b) suma și media aritmetică a numerelor pozitive;
  - c) suma și media aritmetică a numerelor negative.
- ❹ Elaborați un program care citește de la tastatură  $n$  caractere și afișează pe ecran:
  - a) numărul cifrelor zecimale citite;
  - b) numărul cifrelor pare;
  - c) numărul cifrelor impare;
  - d) numărul literelor citite;
  - e) numărul vocalelor;
  - f) numărul consoanelor.

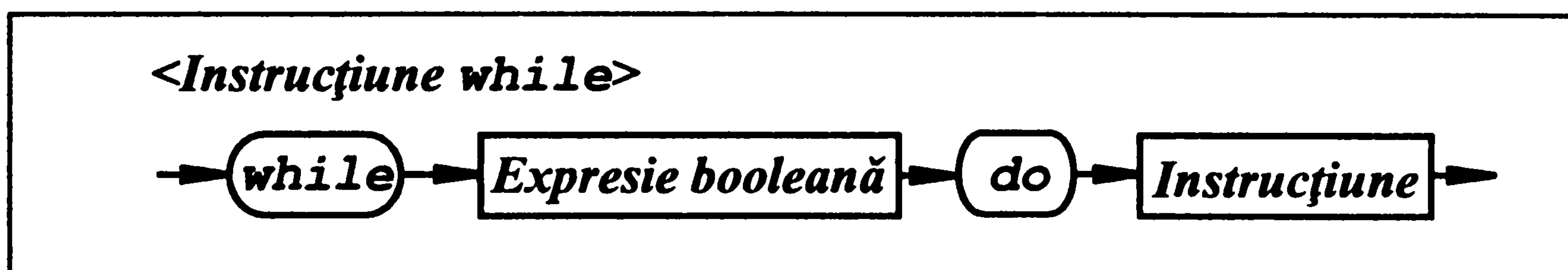
Caracterele introduse se separă prin acționarea tastei *<ENTER>*. Sînt admise cifrele zecimale 0, 1, 2, ..., 9 și literele mari *A, B, C, ..., Z* ale alfabetului latin.

### 3.14. INSTRUCȚIUNEA WHILE

Instrucțiunea **while** conține o expresie booleană care controlează execuția repetată a altei instrucțiuni. Sintaxa instrucțiunii în studiu este:

*<Instrucțiune while> ::=*  
**while** *<Expresie booleană>* **do** *<Instrucțiune>*

Diagrama sintactică este prezentată în *fig. 3.11*.



*Fig. 3.11.* Diagrama sintactică a instrucțiunii **while**

*Exemple:*

- 1) **while** *x* > 0 **do** *x* := *x* - 1;
- 2) **while** *x* < 3.14 **do**  
**begin**  
*x* := *x* + 0.001;  
 writeln(sin(*x*));  
**end;**
- 3) **while** *p* **do**  
**begin**  
*x* := *x* + 0.001;  
*y* := 10 \* *x*;  
*p* := *y* < 1000;  
**end;**

Instrucțiunea situată după cuvîntul-cheie **do** se execută repetat atîta timp, cît valoarea expresiei booleene este *true*. Dacă expresia booleană ia valoarea *false*, instrucțiunea de după **do** nu se mai execută. Se recomandă ca expresia booleană să fie cît mai simplă, deoarece ea este evaluată la fiecare iterație.

În mod obișnuit, instrucțiunea **while** se utilizează pentru organizarea calculelor repetitive cu variabile de control de tip *real*.

În programul ce urmează, instrucțiunea **while** este utilizată pentru afișarea valorilor funcției  $y = 2x$ . Argumentul *x* ia valori de la  $x_1$  la  $x_2$  cu pasul  $\Delta x$ .



```

Program P65;
{ Tabelul funcției $y=2*x$ }
var x, y, x1, x2, deltaX : real;
begin
 write('x1='); readln(x1);
 write('x2='); readln(x2);
 write('deltaX='); readln(deltaX);
 writeln('x':10, 'y':20);
 writeln;
 x:=x1;
 while x<=x2 do
 begin
 y:=2*x;
 writeln(x:20, y:20);
 x:=x+deltaX;
 end;
 readln;
end.

```

Instrucțiunea **while** este deosebit de utilă în situația în care numărul de execuții repetate ale unei secvențe de instrucțiuni e dificil de evaluat.

Pentru exemplificare prezentăm programul P66, care afișează pe ecran media aritmetică a numerelor pozitive citite de la tastatură.

```

Program P66;
{ Media aritmetică a numerelor pozitive
 citite de la tastatură }
var x, suma : real; n : integer;
begin
 n:=0;
 suma:=0;
 writeln('Dați numere pozitive:');
 readln(x);
 while x>0 do
 begin
 n:=n+1;
 suma:=suma+x;
 readln(x);
 end;
 writeln('Ați introdus ', n, ' numere pozitive. ');
 if n>0 then writeln('media=', suma/n)
 else writeln('media=*****');
 readln;
end.

```

Se observă că numărul de execuții repetate ale instrucțiunii compuse **begin ... end** din componența instrucțiunii **while** nu poate fi calculat din timp. Execuția instrucțiunii **while** se termină când utilizatorul introduce un număr  $x \leq 0$ .

## Întrebări și exerciții

- ❶ Cum se execută o instrucțiune **while**?
- ❷ Indicați pe diagramele sintactice din *fig. 3.11* drumurile care corespund instrucțiunilor **while** din programele P65 și P66.
- ❸ Utilizând instrucțiunea **while**, scrieți un program care afișează pe ecran valorile funcției  $y = f(x)$  pentru valori ale argumentului de la  $x_1$  la  $x_2$  cu pasul  $\Delta x$ :

$$\text{a) } y = x^2 ; \qquad \text{b) } y = \begin{cases} \sin x, & x \geq 0 ; \\ \cos x, & x < 0 ; \end{cases}$$

$$\text{c) } y = \begin{cases} \sqrt{x}, & x \geq 0 ; \\ x^2, & x < 0 ; \end{cases} \qquad \text{d) } y = \begin{cases} x^2, & |x| > 5 ; \\ x^3, & |x| \leq 5 ; \end{cases}$$

- ❹ Utilizatorul introduce de la tastatură numere întregi pozitive, separate prin acționarea tastei *<ENTER>*. Sfârșitul secvenței de numere e indicat prin introducerea numărului 0. Scrieți un program care afișează pe ecran:
  - a) suma și media aritmetică a numerelor pare;
  - b) suma și media aritmetică a numerelor impare.
- ❺ Scrieți un program care afișează pe ecran valorile funcției  $z = f(x, y)$ . Argumentul  $x$  ia valori de la  $x_1$  la  $x_2$  cu pasul  $\Delta x$ ; argumentul  $y$  ia valori de la  $y_1$  la  $y_2$  cu pasul  $\Delta y$ :

$$\text{a) } z = \begin{cases} x + y, & x \geq y ; \\ x - y, & x < y ; \end{cases} \qquad \text{b) } z = \begin{cases} \sin(x + y), & x > 2y ; \\ \cos(x + y), & x \leq 2y ; \end{cases}$$
$$\text{c) } z = \begin{cases} x^2 + y^2, & x + y > 6 ; \\ x^3 - y^3, & x + y \leq 6 ; \end{cases} \qquad \text{d) } z = \begin{cases} x + y, & x > 8y ; \\ \text{abs}(x + y), & x \leq 8y ; \end{cases}$$

*Exemplu:*  $z = x + y$ .

```
Program P67;
{ Tabelul funcției z=x+y }
var x, y, z,
 x1, x2, deltaX,
 y1, y2, deltaY : real;
begin
 write('x1=');
 readln(x1);
 write('x2=');
 readln(x2);
 write('deltaX=');
 readln(deltaX);
 write('y1=');
 readln(y1);
 write('y2=');
 readln(y2);
 write('deltaY=');
 readln(deltaY);
```

```

 writeln('x':10, 'y':20, 'z':20);
 writeln;
 x:=x1;
while x<=x2 do
begin
 y:=y1;
 while y<=y2 do
 begin
 z:=x+y;
 writeln(x:20, y:20, z:20);
 y:=y+deltaY;
 end;
 x:=x+deltaX;
end;
 readln;
end.

```

#### ⑥ Instrucțiunea repetitivă

**for** i:=i1 **to** i2 **do** writeln(ord(i));

este echivalentă cu secvența de instrucțiuni

```

i:=i1;
while i<=i2 do
begin
 writeln(ord(i));
 i:=succ(i);
end.

```

Scrieți o secvență echivalentă pentru instrucțiunea repetitivă

**for** i:=i1 **downto** i2 **do** writeln(ord(i)).

#### ⑦ Se consideră declarațiile

```

var x1, x2, deltaX : real;
 i, n : integer;

```

Care din secvențele de instrucțiuni ce urmează sînt echivalente?

- a) **x:=x1;**  
**while** x<=x2 **do**  
**begin**  
 writeln(x);  
 x:=x+deltaX;  
**end**;
- b) n:=trunc((x2-x1)/deltaX)+1;  
 x:=x1;  
**for** i:=1 **to** n **do**  
**begin**  
 writeln(x);  
 x:=x+deltaX;  
**end**;



```

c) n:=round((x2-x1)/deltaX)+1;
 x:=x1;
 for i:=1 to n do
 begin
 writeln(x);
 x:=x+deltaX;
 end;

```

Argumentați răspunsul.

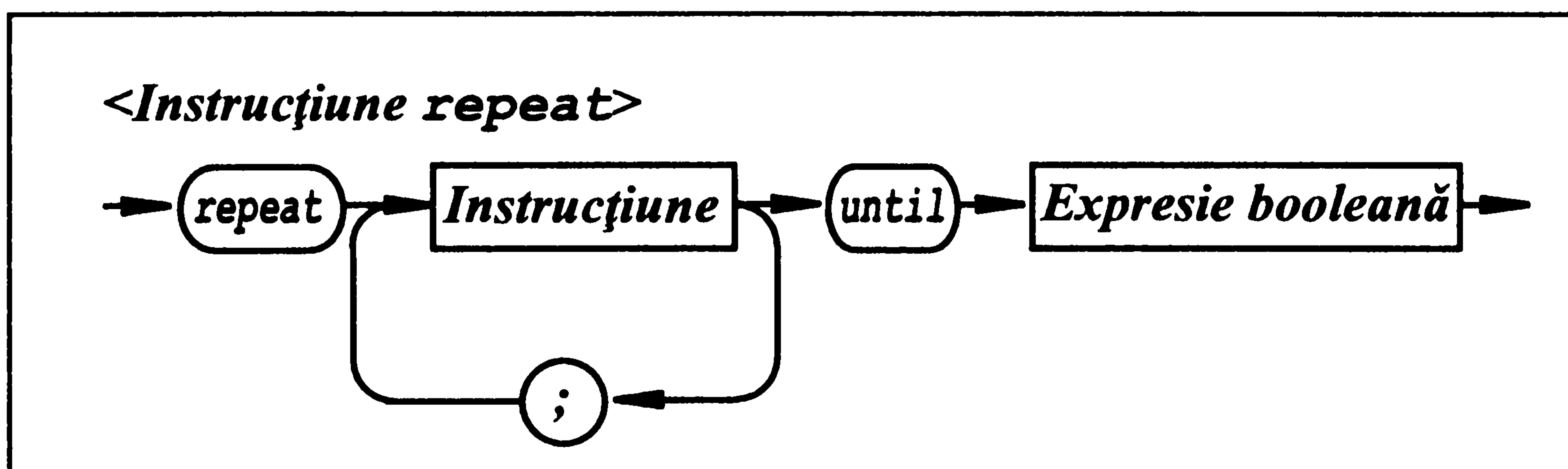
### 3.15. INTRUCȚIUNEA REPEAT

IntruȚiunea **repeat** indică repetarea unei secvenȚe de intruȚiuni în funcȚie de valoarea unei expresii booleene.

Sintaxa intruȚiunii este:

*<IntruȚiune repeat> ::= repeat <IntruȚiune>  
 {;<IntruȚiune>} until <Expresie booleană>*

Diagrama sintactică este prezentată în *fig. 3.12*.



*Fig. 3.12.* Diagrama sintactică a intruȚiunii **repeat**

*Exemple:*

- 1) **repeat** x:=x-1 **until** x<0;
- 2) **repeat**  
     y:=y+delta;  
     writeln(y)  
   **until** y>20.5;
- 3) **repeat**  
     readln(i);  
     writeln(odd(i))  
   **until** i=0;

IntruȚiunile situate între cuvintele-cheie **repeat** și **until** se execută repetat atît timp, cît expresia booleană este falsă. Cînd această expresie devine adevărată, se trece la intruȚiunea următoare. Evident, intruȚiunile aflate între **repeat** și **until** vor fi executate

cel puțin o dată, deoarece evaluarea expresiei logice are loc după ce s-a executat această secvență.

În mod obișnuit, instrucțiunea **repeat** se utilizează în locul instrucțiunii **while** atunci când evaluarea expresiei care controlează repetiția se face după executarea secvenței de repetat.

Programul ce urmează afișează la ecran paritatea numerelor întregi citite de la tastatură.

```
Program P68;
{ Paritatea numerelor citite de la tastatură }
var i : integer;
begin
 writeln('Dați numere întregi:');
 repeat
 readln(i);
 if odd(i) then writeln(i:6, ' - număr impar')
 else writeln(i:6, ' - număr par');
 until i=0;
 readln;
end.
```

Execuția instrucțiunii **repeat** se termină când utilizatorul introduce  $i=0$ .

Următorul program calculează valoarea  $y = \sqrt{x}$ ,  $x > 0$ , utilizând relația recurentă:

$$y_i = \frac{y_{i-1}}{2} + \frac{x}{2y_{i-1}}, \quad y_1 = 1.$$

Iterațiile se termină când pentru aproximările succesive  $y_{i-1}$  și  $y_i$  se respectă condiția

$$|y_{i-1} - y_i| \leq \varepsilon.$$

```
Program P69;
{ Calcularea rădăcinii pătrate }
var x, epsilon, y1, y2: real;
begin
 write('x='); readln(x);
 write('epsilon='); readln(epsilon);
 y2:=1;
 repeat
 y1:=y2; y2:=y1/2+x/(2*y1);
 writeln(y2);
 until abs(y1-y2)<=epsilon;
 readln;
end.
```

Parametrul  $\varepsilon$  specifică precizia cu care se va calcula  $\sqrt{x}$ . De exemplu, pentru  $x = 2$  și  $\varepsilon = 0.01$  programul P69 va afișa:

```
1.500000000000E+00
1.41666666667E+00
1.4142156863E+00,
```

iar pentru  $x = 2$  și  $\varepsilon = 0.0000000001$ :

```
1.500000000000E+00
1.41666666667E+00
1.4142156863E+00
1.4142135624E+00
1.4142135624E+00.
```

Din exemplele studiate se observă că instrucțiunea **repeat** este utilă în situația în care numărul de executări repetate ale unei secvențe de instrucțiuni este dificil de evaluat.

## Întrebări și exerciții

- ❶ Cum se execută o instrucțiune **repeat**?
- ❷ Indicați pe diagramele sintactice din *fig. 3.12* drumurile care corespund instrucțiunilor **repeat** din programele P68 și P69.
- ❸ Se consideră instrucțiunile:

a) **repeat**  
    <Instrucțiune 1>  
    <Instrucțiune 2>  
    ...  
    <Instrucțiune n>  
**until** p;

b) **while not p do**  
    **begin**  
        <Instrucțiune 1>  
        <Instrucțiune 2>  
        ...  
        <Instrucțiune n>  
    **end.**

Sînt oare echivalente aceste instrucțiuni? Argumentați răspunsul.

- ❹ Elaborați un program care citește de la tastatură o secvență de caractere și afișează pe ecran:
  - a) numărul cifrelor zecimale citite;
  - b) numărul cifrelor pare;
  - c) numărul cifrelor impare.

Caracterele introduse se separă prin acționarea tastei <ENTER>. Sînt admise cifrele zecimale 0, 1, 2, ..., 9 și caracterul \* care indică sfîrșitul secvenței.

- ❺ Elaborați un program pentru calcularea valorii  $y = \sqrt[n]{x}$  cu ajutorul relației recurente:



$$y_i = \frac{2y_{i-1}}{3} + \frac{x}{3y_{i-1}^2}, \quad y_1 = 1.$$

⑥ Este oare echivalentă instrucțiunea

**for**  $i := i_1$  **to**  $i_2$  **do** `writeln(ord(i))`

cu secvența de instrucțiuni

$i := i_1$ ;

**repeat**

`writeln(ord(i));`

$i := \text{succ}(i)$

**until**  $i > i_2$ ;

Argumentați răspunsul.

⑦ Scrieți un program care afișează pe ecran valorile funcției  $y = f(x)$ . Argumentul  $x$  ia valori de la  $x_1$  la  $x_2$  cu pasul  $\Delta x$ . Ciclul se va organiza cu ajutorul instrucțiunii **repeat**.

$$\begin{array}{ll} \text{a) } y = x^4; & \text{c) } y = \begin{cases} 3x, & x \geq 4; \\ \frac{x}{3}, & x < 4; \end{cases} \\ \text{b) } y = \sqrt[4]{x}; & \text{d) } y = \begin{cases} 16 - \frac{x}{3}, & x > 0; \\ 26 + x^2, & x \leq 0. \end{cases} \end{array}$$

⑧ Elaborați un program care citește de pe tastatură o secvență de caractere și afișează pe ecran:

- a) numărul literelor citite;
- b) numărul literelor mari;
- c) numărul literelor mici.

Caracterele introduse se separă prin acționarea tastei `<ENTER>`. Sînt admise literele mari și mici ale alfabetului latin și caracterul `*` care indică sfîrșitul secvenței.

### 3.16. INSTRUCȚIUNEA GOTO

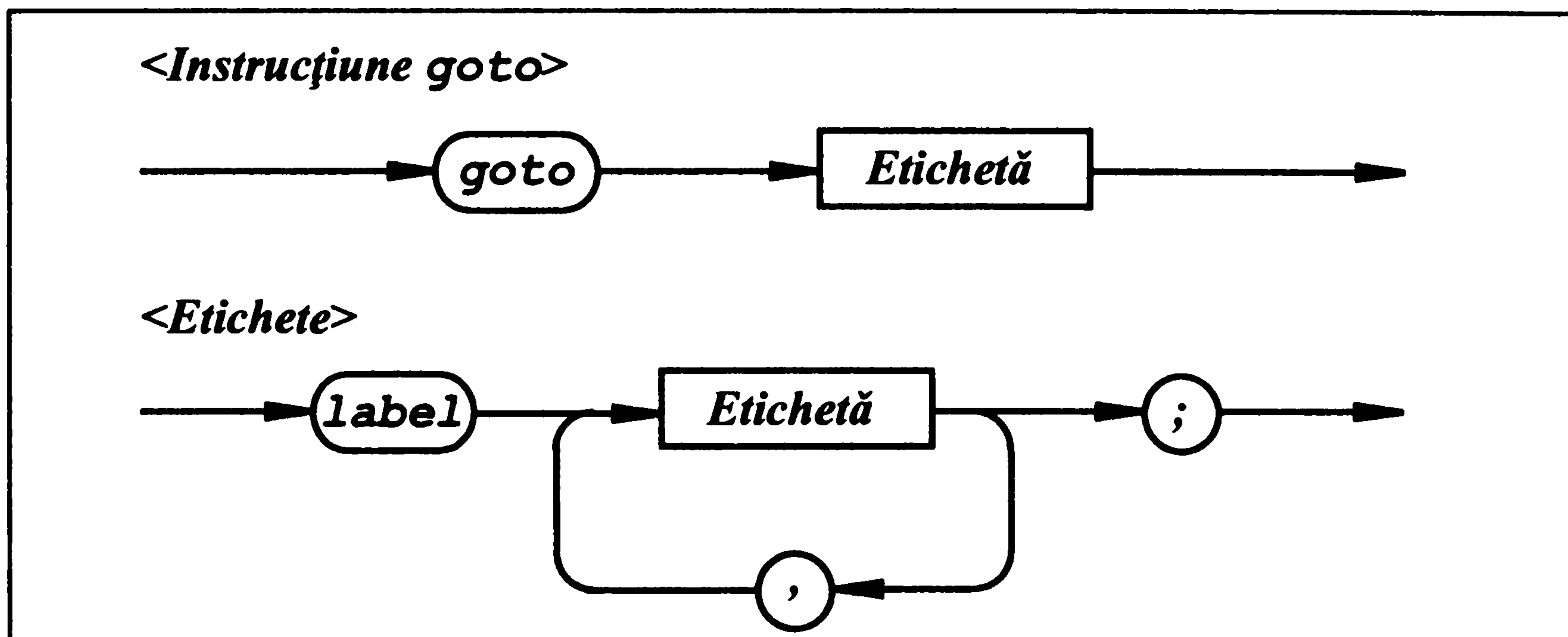
Instrucțiunile unui program sînt executate secvențial, așa cum apar scrise în textul programului. Instrucțiunea de salt necondiționat **goto** oferă posibilitatea de a întrerupe această secvență și de a relua execuția dintr-un alt loc al programului. Instrucțiunea în studiu are sintaxa:

*<Instrucțiune goto> ::= goto <Etichetă>*

Amintim că eticheta este un număr întreg fără semn care prefixează o instrucțiune a programului (fig. 3.1). Etichetele unui program sînt listate în partea declarativă a programului după cuvîntul-cheie **label**. Sintaxa acestei declarații este:

$\langle Etichete \rangle ::= \text{label } \langle Etichetă \rangle \{, \langle Etichetă \rangle \};$

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *fig. 3.13*. De reținut că declararea prin **label** a etichetelor este obligatorie.



*Fig. 3.13.* Diagramele sintactice  $\langle Instrucțiune \text{ goto} \rangle$  și  $\langle Etichete \rangle$

Execuția instrucțiunii **goto** are ca efect transferul controlului la instrucțiunea prefixată de eticheta respectivă.

Pentru exemplificare prezentăm programul P70 care calculează valoarea funcției

$$y = \begin{cases} x, & x \geq 0; \\ 2x, & x < 0. \end{cases}$$

Valorile argumentului  $x$  se citesc de la tastatură.

```

Program P70;
{ Execuția instrucțiunii goto }
label 1, 2;
var x, y : real;
begin
 write('x='); readln(x);
 if x>=0 then goto 1;
 y:=2*x;
 writeln('x<0, y=', y);
 goto 2;
1 : y:=x;
 writeln('x>=0, y=', y);
2 : readln;
end.

```

Dacă utilizatorul tastează o valoare  $x \geq 0$ , se va executa instrucțiunea **goto** 1 și controlul va trece la instrucțiunea de atribuire

1:  $y := x$

După aceasta se vor executa instrucțiunile

```
writeln('x>=0 , y=', y);
2: readln
```

Dacă utilizatorul tastează o valoare  $x < 0$ , se execută instrucțiunile

```
y:= 2*x;
writeln('x<0 , y=', y);
goto 2
```

Ultima instrucțiune va transfera controlul instrucțiunii

```
2: readln
```

Etichetele și instrucțiunile unui program trebuie să respecte următoarele reguli:

- 1) orice etichetă trebuie să fie declarată cu ajutorul cuvântului-cheie **label**;
- 2) orice etichetă trebuie să prefixeze o singură instrucțiune;
- 3) este interzis saltul din afara unei instrucțiuni structurate (**if**, **for**, **while**, ... ș.a.) în interiorul ei.

*Exemple:*

```
if i>5 then 1: writeln('i>5')
else writeln('i<=5');
...
goto 1; { Eroare}
for i:=1 to 10 do
begin
10: writeln('i=', i);
end;
...
goto 10; { Eroare}
```

În lipsa lui **goto** instrucțiunile unui program sînt executate în ordinea în care sînt scrise. Prin urmare, instrucțiunile **goto** violează concordanța dintre textul programului și ordinea de execuție a instrucțiunilor. Acest fapt complică elaborarea, verificarea și depanarea programelor. În consecință, folosirea instrucțiunii **goto** nu este recomandată.

De exemplu, programul P70 poate fi refăcut după cum urmează:

```
Program P71;
{ Excluderea instrucțiunii goto din programul P70 }
var x, y : real;
begin
 write('x=');
 readln(x);
 if x>=0 then
 begin
 y:=x;
 writeln('x>=0, y=', y);
```



```

end
else
begin
 y:=2*x;
 writeln('x<0, y=', y);
end;
readln;
end.

```

De regulă, instrucțiunea **goto** se utilizează în cazuri extraordinare, de exemplu, pentru a mări viteza de derulare sau pentru a micșora lungimea unui program.

### Întrebări și exerciții

- ❶ Care este destinația instrucțiunii **goto**?
- ❷ Indicați pe diagramele sintactice din *fig. 3.13* drumurile care corespund etichetelor și instrucțiunilor **goto** din programul P70.
- ❸ Transcrieți fără a utiliza instrucțiunea **goto**:

```

Program P72;
{ Afișarea formulelor de salut }
label 1, 2, 3;
var i : 6..23;
begin
 write('Cît e ora?'); readln(i);
 if i>12 then goto 1;
 writeln('Bună dimineața!');
 goto 3;
1: if i>17 then goto 2;
 writeln('Bună ziua!');
 goto 3;
2: writeln('Bună seara!');
3: readln;
end.

```

- ❹ Comentați următorul program:

```

Program P73;
{ Eroare }
label 1;
var i : 1..5;
begin
 i:=1;
1: writeln(i);
 i:=i+1;
 goto 1;
end.

```

- ❺ Ce va afișa pe ecran programul ce urmează?

```

Program P74;
 { Eroare }
label 1;
var x: real;
begin
 x:=0;
1: writeln(x);
 x:=x+1e-30;
 goto 1;
end.

```

Amintim că derularea unui program poate fi întreruptă prin acționarea tastelor *<CTRL+C>* sau *<CTRL+BREAK>*.

⑥ Comentați programul ce urmează :

```

Program P75;
 { Eroare }
label 1;
var i : integer;
begin
 i:=1;
 while i<=20 do
 begin
 writeln(i);
1: i:=i+1;
 end; goto 1;
end.

```

### 3.17. GENERALITĂȚI DESPRE STRUCTURA UNUI PROGRAM PASCAL

Un program PASCAL are următoarea structură:

*<Program> ::=*  
     *<Antet program>*  
     *<Corp> .*

**Antetul** specifică numele programului și include, opțional, o listă de parametri formali:

*<Antet program> ::=*  
     **Program** *<Identificator>*  
         [ (*<Identificator>* {, *<Identificator>*}) ];

*Exemple:*

```

Program A1;
Program B6(Intrare);
Program C15(Intrare, Iesire);

```

În mod obișnuit, parametrii formali se folosesc pentru comunicarea programului cu mediul său. Aceștia vor fi studiați mai amănunțit în capitolele următoare.

**Corpul** unui program este compus din partea declarativă și partea executabilă:

$\langle Corp \rangle ::= \langle Declarații \rangle$   
 $\quad \quad \quad \langle Instrucțiune compusă \rangle$

**Partea declarativă** a programului are următoarea sintaxă:

$\langle Declarații \rangle ::=$  [ $\langle Etichete \rangle$ ]  
                  [ $\langle Constante \rangle$ ]  
                  [ $\langle Tipuri \rangle$ ]  
                  [ $\langle Variabile \rangle$ ]  
                  [ $\langle Subprograme \rangle$ ]

Subprogramele vor fi studiate în capitolele următoare.

**Partea executabilă** a unui program este o instrucțiune compusă **begin...end**.

Diagramele sintactice ale unităților gramaticale în studiu sînt prezentate în *fig. 3.14*.

Subliniem faptul că sfîrșitul unui program PASCAL este indicat de simbolul **"."** (**punct**).

Pentru exemplificare prezentăm în continuare programul P76 care calculează lungimea arcului de cerc de  $\alpha$  grade și aria sectorului respectiv.

```
Program P76;
 { Lungimea arcului de cerc
 și aria sectorului respectiv }
label 1, 2;
const Pi=3.141592654;
type grade=0..360;
var alfa : grade;
 raza, lungimea, aria : real;
begin
 write('raza='); readln(raza);
 if raza<0 then goto 1;
 write('alfa='); readln(alfa);
 lungimea:=Pi*raza*alfa/180;
 writeln('lungimea=', lungimea);
 aria:=Pi*sqr(raza)*alfa/360;
 writeln('aria=', aria);
 goto 2;
1: writeln('Eroare: raza<0');
2: readln;
end.
```



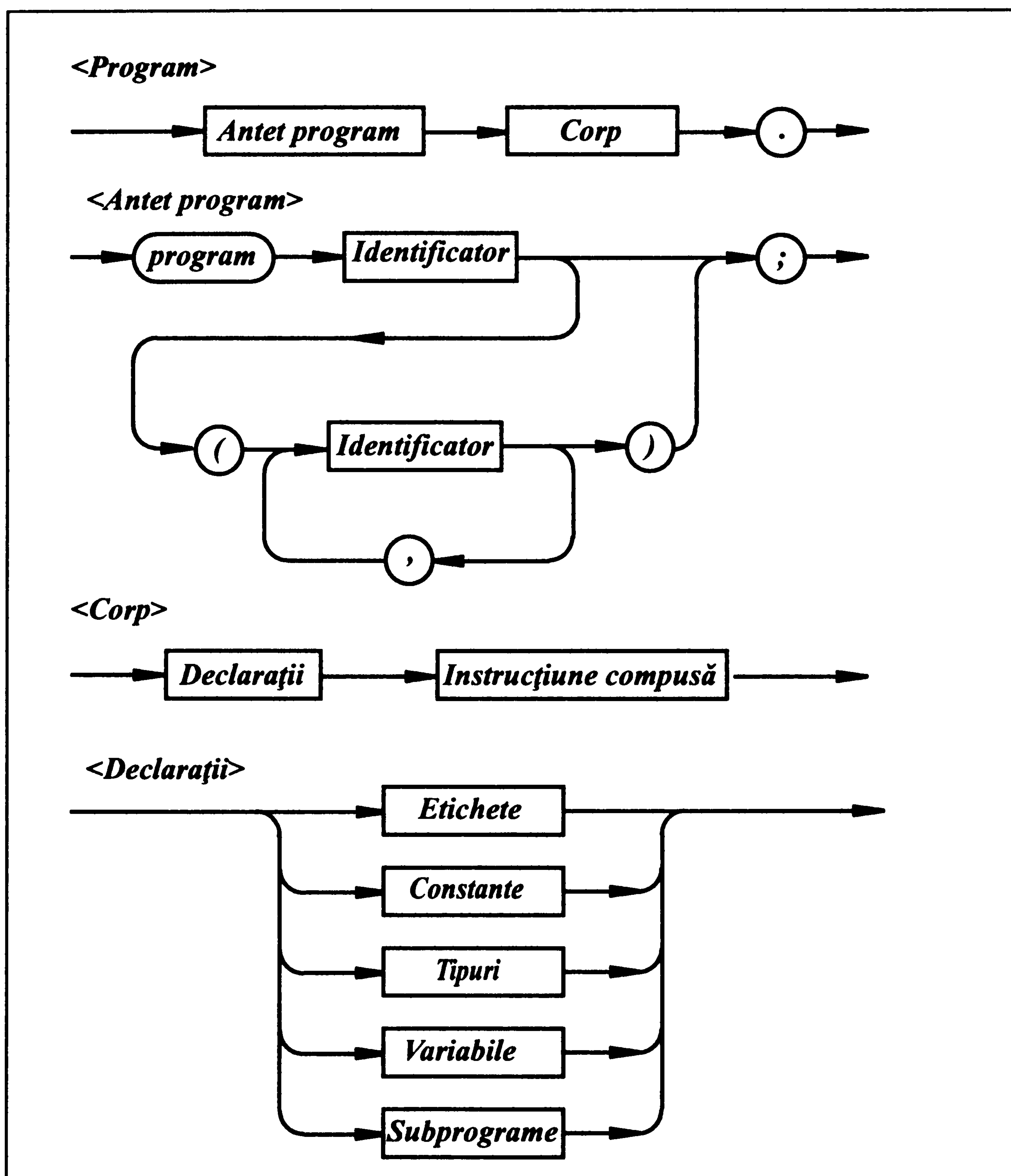


Fig. 3.14. Structura unui program PASCAL

## Întrebări și exerciții

- ❶ Care este destinația antetului de program? Cum se indică sfârșitul unui program PASCAL?
- ❷ Indicați pe diagramele sintactice din *fig. 3.14* drumurile care corespund unităților gramaticale ale programului P76.
- ❸ Transcrieți programul P76 fără a utiliza instrucțiunea **goto**. Indicați partea declarativă și partea executabilă a programului elaborat.
- ❹ Care este destinația parametrilor dintr-un antet de program?

## Capitolul 4

.....

# TIPURI DE DATE STRUCTURATE

### 4.1. TIPURI DE DATE *TABLOU* (ARRAY)

Mulțimea de valori ale unui tip de date **array** este constituită din tablouri (tabele). Tablourile sînt formate dintr-un număr fixat de componente de același tip, denumit **tip de bază**. Referirea componentelor se face cu ajutorul unui **indice**.

Un tip de date *tablou* se definește printr-o construcție de forma

**type** <*Nume tip*> = **array** [*T1*] **of** *T2*;

unde *T1* este tipul indicelui care trebuie să fie ordinal, iar *T2* este tipul componentelor (tipul de bază) care poate fi un tip oarecare.

*Exemple:*

```
1) type Vector = array[1..5] of real;
 var x: Vector;
2) type Zi= (L, Ma, Mi, J, V, S, D);
 Venit= array[Zi] of real;
 var v: Venit;
 z: Zi;
3) type Ora= 0..23;
 Grade= -40..40;
 Temperatura= array[Ora] of Grade;
 var t: Temperatura;
 h: Ora;
```

Structura datelor din exemplele în studiu este prezentată în *fig. 4.1*.

Fiecare componentă a unei variabile de tip tablou poate fi specificată explicit, prin numele variabilei urmat de indicele respectiv încadrat de paranteze pătrate.

*Exemple:*

```
x[1] , x[4] ;
v[L] , v [Ma] , v[J] ;
t[0] , t[15] , t[23] ;
v[z] , t[h] .
```

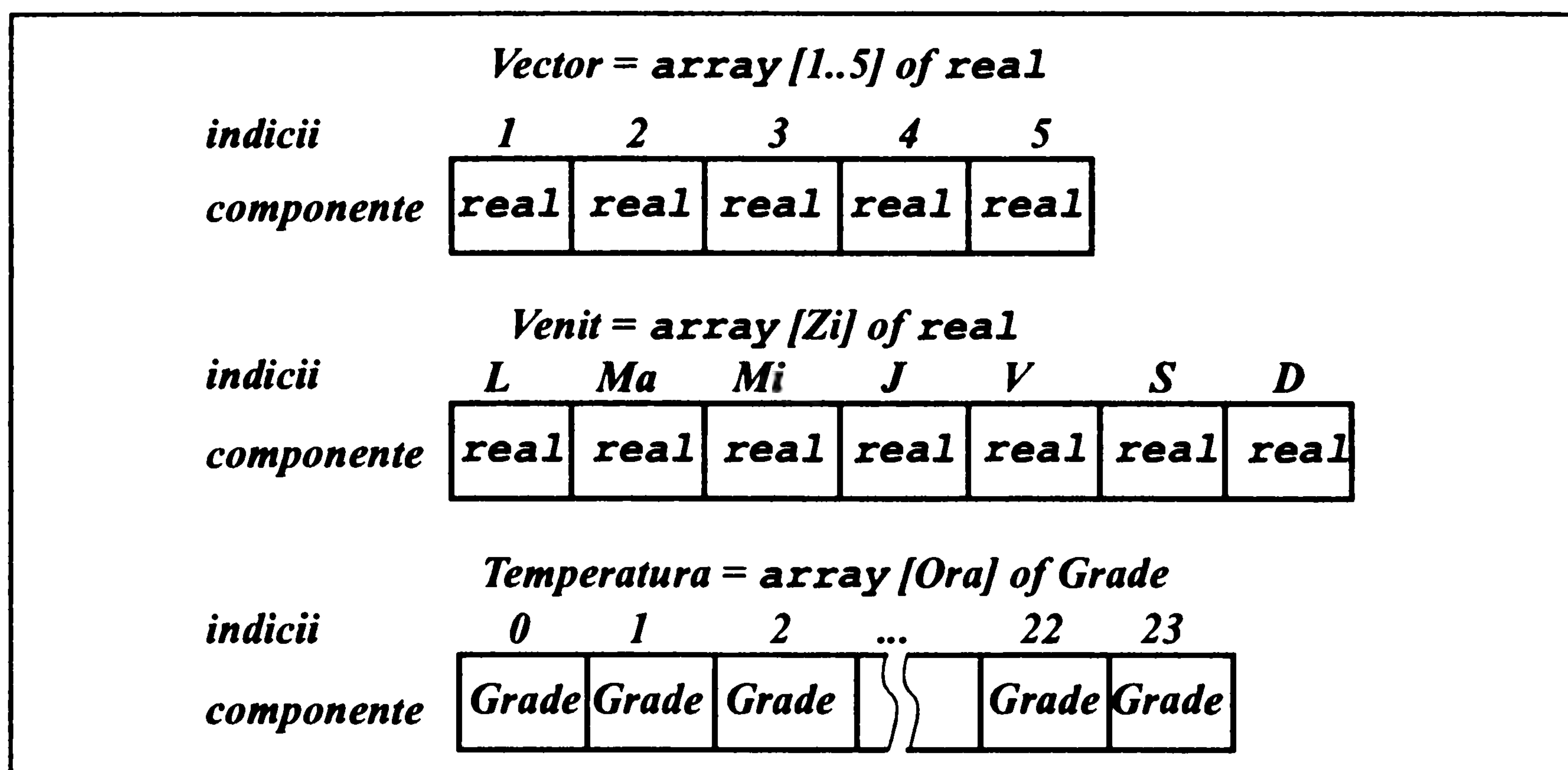


Fig. 4.1. Structura datelor de tip Vector, Venit și Temperatura

Asupra componentelor datelor de tip *tablou* se pot efectua toate operațiile admise de tipul de bază respectiv. Programul ce urmează afișează pe ecran suma componentelor variabilei *x* de tip Vector. Valorile componentelor *x* [ 1 ] , *x* [ 2 ] , ..., *x* [ 5 ] se citesc de la tastatură.

```

Program P77;
{ Suma componentelor variabilei x de tip Vector }
type Vector=array[1..5] of real;
var x : Vector;
 i : integer;
 s : real;
begin
 writeln('Dați 5 numere:');
 for i:=1 to 5 do readln(x[i]);
 writeln('Ați introdus:');
 for i:=1 to 5 do writeln(x[i]);
 s:=0;
 for i:=1 to 5 do s:=s+x[i] ;
 writeln('Suma=', s);
 readln;
end.

```

Pentru a extinde aria de aplicare a unui program, se recomandă ca numărul de componente ale datelor de tip **array** să fie specificate prin constante.

De exemplu, programul P77 poate fi modificat pentru a însuma *n* numere reale,  $n \leq 100$ :

```

Program P78;
{ Extinderea domeniului de aplicare a programului P77 }
const nmax=100;

```



```

type Vector=array[1..nmax] of real;
var x : Vector;
 n : 1..nmax;
 i : integer;
 s : real;
begin
 write('n='); readln(n);
 writeln('Dați ', n, ' numere:');
 for i:=1 to n do readln(x[i]);
 writeln('Ați introdus:');
 for i:=1 to n do writeln(x[i]);
 s:=0;
 for i:=1 to n do s:=s+x[i];
 writeln('Suma=', s);
 readln;
end.

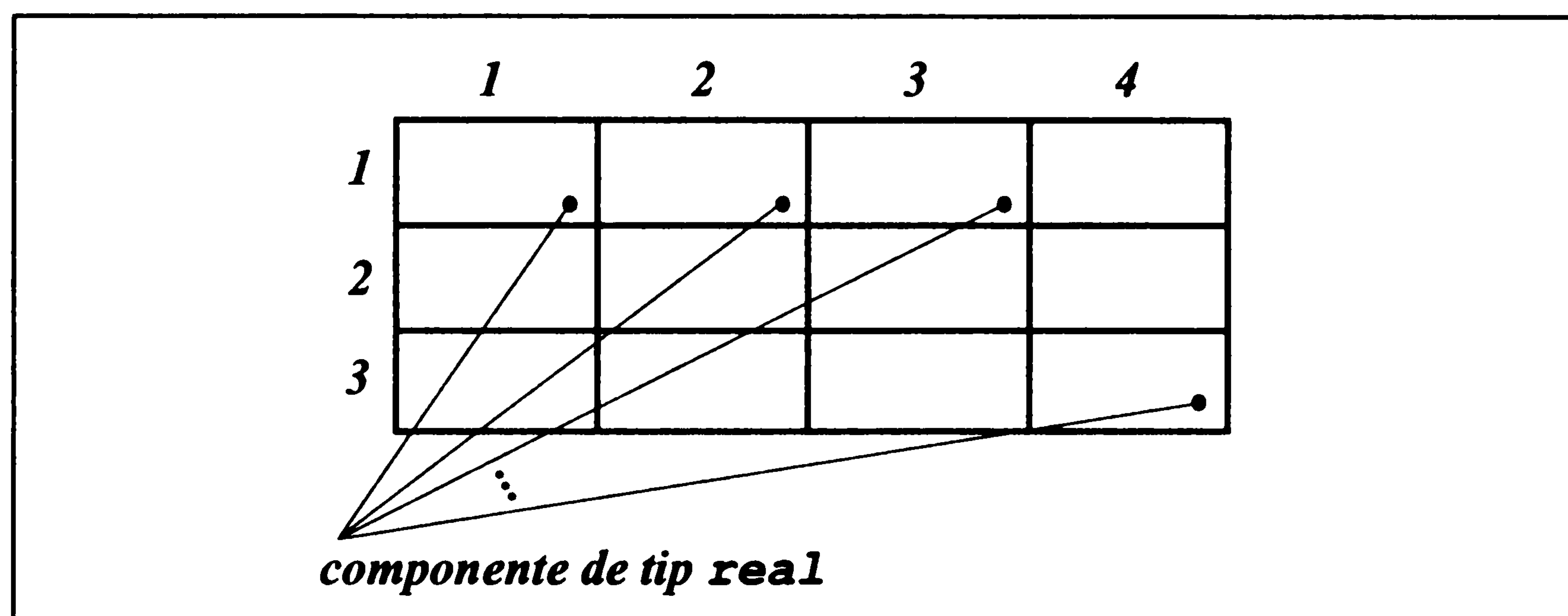
```

Tablourile bidimensionale se definesc cu ajutorul construcției

**type** <Nume tip> = **array**[ *T1*, *T2*] **of** *T3*,

unde *T1* și *T2* specifică tipul indicilor, iar *T3* tipul componentelor.

Pentru exemplificare, în *fig. 4.2.* este prezentată structura datelor tipului:



*Fig. 4.2.* Structura datelor de tip Matrice

Matrice = **array**[ 1..3, 1..4] **of** real.

Componentele unei variabile de tip *tablou* bidimensional se specifică explicit prin numele variabilei urmat de indicii respectivi separați prin virgulă și încadrați de paranteze pătrate.

De exemplu, în prezența declarației

**var** m : Matrice;

notația *m* [ 1, 1] specifică componenta din linia 1, coloana 1 (vezi *fig. 4.2.*); notația *m* [ 1, 2] specifică componenta din linia 1, coloana 2; notația *m* [ *i*, *j*] specifică componenta din linia *i*, coloana *j*.

Programul ce urmează afișează pe ecran suma componentelor variabilei *m* de tip *Matrice*. Valorile componentelor *m* [ 1, 1], *m* [ 1, 2] , ..., *m* [ 3, 4] se citesc de la tastatură.

```
Program P79;
{ Suma componentelor variabilei m de tip Matrice }
type Matrice=array[1..3, 1..4] of real;
var m : Matrice;
 i, j : integer;
 s : real;
begin
 writeln('Dați componentele m[i,j]:');
 for i:=1 to 3 do
 for j:=1 to 4 do
 begin
 write('m[', i, ', ', j, ']=');
 readln(m[i,j]);
 end;
 writeln('Ați introdus:');
 for i:=1 to 3 do
 begin
 for j:=1 to 4 do
 write(m[i,j]);
 writeln;
 end;
 s:=0;
 for i:=1 to 3 do
 for j:=1 to 4 do
 s:=s+m[i,j] ;
 writeln('Suma=', s);
 readln;
end.
```

În general, un tip **tablou** *n*-dimensional (*n* = 1, 2, 3 ș.a.m.d) se definește cu ajutorul diagramelor sintactice din *fig. 4.3*. Atributul **packed** (împachetat) indică cerința de optimizare a spațiului de memorie pentru elementele tipului **array**. Menționăm că în majoritatea compilatoarelor actuale utilizarea acestui atribut nu are nici un efect, întrucât optimizarea se efectuează în mod automat.

Fiind date două variabile de tip *tablou* de același tip, numele variabilelor pot apărea într-o instrucțiune de atribuire. Această atribuire înseamnă copierea tuturor componentelor din membrul drept în membrul stâng.

De exemplu, în prezența declarațiilor

```
var a, b : Matrice;
```

instrucțiunea

$a := b$

este corectă.

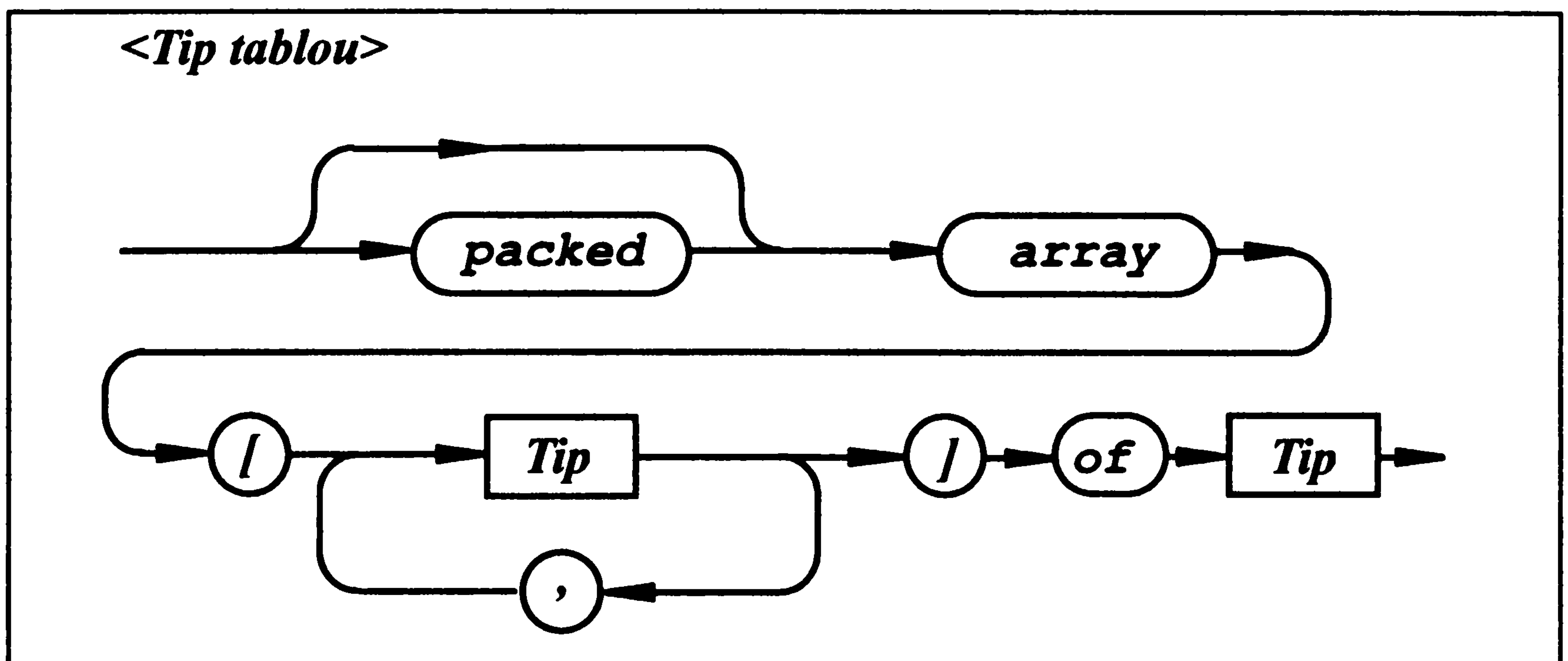


Fig. 4.3. Diagrama sintactică <Tip tablou>

În exemplele de mai sus tipul de bază (tipul componentelor) a fost de fiecare dată un tip simplu. Deoarece tipul de bază poate fi, în general, un tip arbitrar, devine posibilă definirea tablourilor cu componente de tip structurat. Considerăm acum un exemplu în care tipul de bază este el însuși un tip **array**:

```
Type Linie=array[1..4] of real;
 Tabel=array[1..3] of Linie;
var L : Linie;
 T : Tabel;
 x : real;
```

Variabila T este formată din 3 componente: T[ 1] , T[ 2] și T[ 3] de tipul Linie.

Variabila L este formată din 4 componente: L[ 1] , L[ 2] , L[ 3] și L[ 4] de tipul real.

Prin urmare, atribuirile

```
L [1] :=x; x:=L [3] ; T [2] :=L; L:=T [1]
```

sînt corecte.

Elementele variabilei T pot fi specificate prin T[ i][ j] sau prescurtat T[ i, j] . Aici i indică numărul componentei de tip Linie în cadrul variabilei T, iar j — numărul componentei de tip real în cadrul componentei T[ i] de tip Linie.

Subliniem faptul că declarațiile de forma

**array**[ T<sub>1</sub> , T<sub>2</sub>] **of** T<sub>3</sub>

și

**array**[ T<sub>1</sub>] **of** **array**[ T<sub>2</sub>] **of** T<sub>3</sub>



definesc tipuri distincte de date. Prima declarație definește tablouri bidimensionale cu componente de tipul  $T_3$ . A doua declarație definește tablouri unidimensionale cu componente de tipul **array**[  $T_2$ ] **of**  $T_3$ .

În programele PASCAL tablourile se utilizează pentru a grupa sub un singur nume mai multe variabile cu caracteristici identice.

## Întrebări și exerciții

❶ Precizați tipul indicilor și tipul componentelor din următoarele declarații:

```
type P=array[1..5] of integer;
 Culoare=(Galben, Verde, Albastru, Violet);
 R=array[Culoare] of real;
 S=array[Culoare, 1..3] of boolean;
 T=array[boolean] of Culoare;
```

Reprezentați structura datelor de tipul P, R, S și T pe un desen (fig. 4.1 și 4.2).

❷ Indicați pe diagrama sintactică din fig. 4.3 drumurile care corespund declarațiilor din exercițiul 1.

❸ Scrieți formulele metalingvistice care corespund diagramei sintactice <Tip tablou> din fig. 4.3.

❹ Se consideră declarațiile:

```
type Vector= array[1..5] of real;
var x, y : Vector;
```

Scrieți expresia aritmetică a cărei valoare este:

- suma primelor trei componente ale variabilei x;
- suma tuturor componentelor variabilei y;
- produsul tuturor componentelor variabilei x;
- valoarea absolută a componentei a treia a variabilei y;
- suma primelor componente ale variabilelor x și y.

❺ Se consideră declarațiile

```
type Zi= (L, Ma, Mi, J, V, S, D);
 Venit= array[Zi] of real;
var v : Venit;
```

Componentele variabilei v reprezintă venitul zilnic al unei întreprinderi. Elaborați un program care:

- calculează venitul săptămînal al întreprinderii;
- calculează media venitului zilnic;
- indică ziua în care s-a obținut cel mai mare venit;
- indică ziua cu venitul cel mai mic.

❻ Se consideră declarațiile:

```
type Ora= 0..23;
 Grade= -40..40;
 Temperatura= array[Ora] of Grade;
var t: Temperatura;
```

Componentele variabilei *t* reprezintă temperaturile măsurate din oră în oră pe parcursul a 24 de ore. Elaborați un program care:

- a) calculează temperatura medie;
- b) indică maximul și minimul temperaturii;
- c) indică ora (orele) la care s-a înregistrat temperatura maximă;
- d) indică ora (orele) la care s-a înregistrat temperatura minimă.

⑦ Se consideră declarațiile

```
type Oras = (Chisinau, Orhei, Balti,
 Tighina, Tiraspol);
 Zi = (L, Ma, Mi, J, V, S, D);
 Consum = array[Oras, Zi] of real;
var c : Consum;
 or : Oras;
 z : Zi;
```

Componenta *C[or, z]* a variabilei *C* reprezintă consumul de energie electrică a orașului *or* în ziua *z*. Elaborați un program care:

- a) calculează energia electrică consumată de fiecare oraș pe parcursul unei săptămîni;
- b) calculează energia electrică consumată zilnic de orașele în studiu;
- c) indică orașul cu un consum săptămînal maxim;
- d) indică orașul cu un consum săptămînal minim;
- e) indică ziua în care se consumă cea mai multă energie electrică;
- f) indică ziua cu un consum minim.

⑧ Se consideră declarațiile:

```
type Vector = array[1..5] of real;
 Matrice= array[1..3,1..4] of real;
 Linie= array[1..4] of real;
 Tabel= array[1..3] of Linie;
var V : Vector;
 M : Matrice;
 L : Linie;
 T : Tabel;
 x : real;
 i : integer;
```

Care din atribuirile ce urmează sînt corecte?

- |                         |                             |
|-------------------------|-----------------------------|
| a) <i>T[3] := T[1];</i> | l) <i>T[2] := V;</i>        |
| b) <i>M := T;</i>       | m) <i>L := T[3];</i>        |
| c) <i>L := V;</i>       | n) <i>T[1,2] := M[1,2];</i> |
| d) <i>L[3] := x;</i>    | o) <i>T[1,2] := M[1,2];</i> |
| e) <i>x := i;</i>       | p) <i>M[1] := 4;</i>        |
| f) <i>i := x;</i>       | q) <i>M[1,3] := L[2];</i>   |
| g) <i>L[3] := i;</i>    | r) <i>x := T[1][2];</i>     |
| h) <i>i := M[1,2];</i>  | s) <i>x := M[1];</i>        |
| i) <i>x := V[4];</i>    | t) <i>L := M[1];</i>        |

- j)  $L[3] := V[4]$  ;                      u)  $V[5] := M[3, 4]$  .  
 k)  $T[1] := 4$  ;

- ⑨ Utilizând un tip de date *tablou*, elaborați un program care realizează algoritmul lui Eratostene pentru calcularea numerelor prime mai mici decât un număr natural dat  $n$  ( $n \leq 200$ ).

## 4.2. TIPURI DE DATE ȘIR DE CARACTERE

În limbajul-standard tipul de date *șir de caractere* reprezintă un caz special al tipului **array** și se definește printr-o construcție de forma

$\langle \text{Nume tip} \rangle ::= \text{packed array} [1..n] \text{ of char};$

Mulțimea de valori ale tipului de date în studiu este formată din toate șirurile ce conțin exact  $n$  caractere.

*Exemplu:*

```

Program P80;
{ Șiruri de caractere de lungime constantă }
type Nume=packed array[1..8] of char;
 Prenume=packed array[1..5] of char;
var N : Nume;
 P : Prenume;
begin
 N:='Munteanu';
 P:='Mihai';
 writeln(N);
 writeln(P);
 readln;
end.

```

Rezultatul afișat pe ecran:

Munteanu  
Mihai

Întrucât șirurile de lungime diferită aparțin unor tipuri distincte de date, în cadrul programului P80 nu sînt admise atribuiri de genul:

```

N:= 'Olaru';
P:= 'Ion';

```

În astfel de cazuri, programatorul va completa datele respective cu spațiul pînă la numărul stabilit de caractere  $n$ , de exemplu:

```

N:= 'Olaru ';
P:= 'Ion ';

```



Valorile unei variabile  $v$  de tip **packed array** [ 1..n] of char pot fi introduse de la tastatură numai prin citirea separată a componentelor respective:

```
Read(v[1]); read(v[2]); ...; read(v[n]) .
```

În schimb, o astfel de valoare poate fi afișată în totalitatea ei printr-un singur apel `write(v)` sau `writeln(v)`.

Subliniem faptul că șirurile de caractere de tip **packed array** [ 1..n] of char conțin exact  $n$  caractere, adică sînt **șiruri de lungime** constantă. Evident, lungimea lor nu poate fi modificată pe parcursul derulării programului respectiv. Acest fapt complică elaborarea programelor destinate prelucrării unor șiruri arbitrare de caractere.

Pentru a înlătura acest neajuns, versiunile actuale ale limbajului permit utilizarea șirurilor de caractere de lungime variabilă.

În Turbo PASCAL un tip de date **șir de caractere de lungime variabilă** se declară printr-o construcție de forma:

```
<Nume tip> = string;
```

sau

```
<Nume tip> = string[$nmax$] ;
```

unde  $nmax$  este lungimea maximă pe care o pot avea șirurile respective. În lipsa parametrului  $nmax$  lungimea maximă se stabilește implicit, în mod obișnuit — 255 de caractere.

Asupra șirurilor de tip **string** se poate efectua operația de concatenare (juxtapunere), notată prin "+". Lungimea curentă a unei valori  $v$  de tip **string** poate fi aflată cu ajutorul funcției predefinite `length(v)` care returnează o valoare de tip **integer**. Indiferent de lungime, toate șirurile de caractere de tip **string** sînt compatibile.

*Exemplu:*

```
Program P81;
{ Șiruri de caractere de lungime variabilă }
type Nume=string[8] ;
 Prenume=string[5] ;
 NumePrenume=string;
var N : Nume;
 P : Prenume;
 NP : NumePrenume;
 L : integer;
begin
 N:='Munteanu'; L:=length(N); writeln(N, L:4);
 P:='Mihai'; L:=length(P); writeln(P, L:4);
 NP:=N+' '+P; L:=length(NP); writeln(NP, L:4);
 N:='Olaru'; L:=length(N); writeln(N, L:4);
 P:='Ion'; L:=length(P); writeln(P, L:4);
```

```

NP:=N+' '+P; L:=length(NP); writeln(NP, L:4);
readln;
end.

```

Rezultatele afișate pe ecran:

```

Munteanu 8
Mihai 5
Munteanu Mihai 14
Olaru 5
Ion 3
Olaru Ion 9

```

Se observă că pe parcursul derulării programului în studiu lungimea șirurilor de caractere N, P și NP se schimbă.

Asupra șirurilor de caractere sînt admise operațiile relaționale <, <=, =, >=, >, <>. Șirurile se compară componentă cu componentă de la stînga la dreapta în conformitate cu ordonarea caracterelor în tipul de date char (vezi paragraful 2.5). Ambii operanzi trebuie să fie de tip packed array [1..n] of char cu același număr de componente sau de tip **string**. Evident, operanzii de tip **string** pot avea lungimi arbitrare.

De exemplu, rezultatul operației 'AC' < 'BA' este true, iar rezultatul operației 'AAAAC' < 'AAAAB' este false.

O variabilă de tip *șir de caractere* poate fi folosită fie în totalitatea ei, fie parțial, prin referirea unui caracter din șir.

De exemplu, pentru P='Mihai' avem P [ 1] = 'M', P [ 2] = 'i', P [ 3] = 'h' ș.a.m.d. După executarea secvenței de instrucțiuni

```

P[1] := 'P'; P[2] := 'e'; P[3] := 't';
P[4] := 'r'; P[5] := 'u'

```

variabila P va avea valoarea 'Petru'.

Programul ce urmează citește de la tastatură șiruri arbitrare de caractere și afișează pe ecran numărul de spații în șirul respectiv. Derularea programului se termină după introducerea șirului 'Șfîrșit'.

```

Program P82;
{ Numărul de spații într-un șir de caractere }
var S : string;
 i, j : integer;
begin
 writeln('Dați șiruri de caractere:');
 repeat
 readln(S);
 i:=0;
 for j:=1 to length(S) do
 if S[j]=' ' then i:=i+1;
 writeln('Numărul de spații=', i);

```



```
until S='Sfîrșit';
end.
```

## Întrebări și exerciții

- ❶ Cum se definește un tip de date *șir de caractere*?
- ❷ Ce operații pot fi efectuate asupra șirurilor de caractere?
- ❸ Comentați următorul program:

```
Program P83;
{ Eroare}
var S : packed array[1..5] of char;
begin
 S:='12345';
 writeln(S);
 S:='Sfat';
 writeln(S);
end.
```

- ❹ Elaborați un program care:
  - a) determină numărul de apariții ale caracterului 'A' într-un șir;
  - b) substituie caracterul 'A' prin caracterul '\*';
  - c) radiază din șir caracterul 'B';
  - d) determină numărul de apariții al silabei 'MA' într-un șir;
  - e) substituie silabele 'MA' prin silaba 'TA';
  - f) radiază din șir silaba 'TO';
- ❺ Precizați rezultatul operațiilor relaționale:
  - a) 'B' < 'A';
  - b) 'BB' > 'AA';
  - c) 'BAAAA' < 'AAAAA';
  - d) 'CCCCD' > 'CCCCA';
  - e) 'A A' = 'AA';
  - f) 'BB ' < 'B B';
  - g) 'A' = 'a';
  - h) 'Aa' > 'aA';
  - i) '123' = '321';
  - j) '12345' > '12345'.
- ❻ Se consideră șiruri de caractere formate din literele mari ale alfabetului latin și spații. Elaborați un program care afișează șirurile în studiu după următoarele reguli:
  - fiecare literă de la 'A' pînă la 'Y' se înlocuiește prin următoarea literă din alfabet;
  - fiecare literă 'Z' se înlocuiește prin litera 'A';
  - fiecare spațiu se înlocuiește prin '-'.
- ❼ Elaborați un program care descifrează șirurile cifrate conform regulilor din exercițiul 6.
- ❽ Se consideră  $m, m \leq 100$ , șiruri de caractere formate din literele mici ale alfabetului latin. Elaborați un program care afișează pe ecran șirurile în studiu în ordine alfabetică.
- ❾ Șirul S este compus din cîteva propoziții, fiecare terminîndu-se cu punct, semn de exclamare sau semnul întrebării. Elaborați un program care afișează pe ecran numărul de propoziții din șirul în studiu.



### 4.3. TIPURI DE DATE ARTICOL (RECORD)

Mulțimea de valori ale unui tip de date **record** este constituită din articole (înregistrări). Articolele sînt formate din componente, denumite *cîmpuri*. Spre deosebire de componentele unui tablou, cîmpurile pot fi de tipuri diferite. Fiecare cîmp are un nume (identificator de cîmp).

Un tip de date *articol* se definește printr-o structură de forma

```
type <Nume tip> = record
 <Nume cîmp 1> : T_1 ;
 <Nume cîmp 2> : T_2 ;
 ...
 <Nume cîmp n> : T_n ;
end;
```

unde  $T_1, T_2, \dots, T_n$  specifică tipul cîmpurilor respective. Tipul unui nume de cîmp este arbitrar, astfel un cîmp poate să fie la rîndul lui tot de tip *articol*. Prin urmare, se pot defini tipuri imbricate.

*Exemple:*

```
1) type Elev=record
 Nume : string;
 Prenume : string;
 NotaMedie : real;
end;

Var E1, E2 : Elev;
2) type Punct=record
 x : real; { coordonata x}
 y : real; { coordonata y}
end;

var P1, P2 : Punct;
3) type Triunghi = record
 A : Punct; { vîrful A}
 B : Punct; { vîrful B}
 C : Punct; { vîrful C}
end;

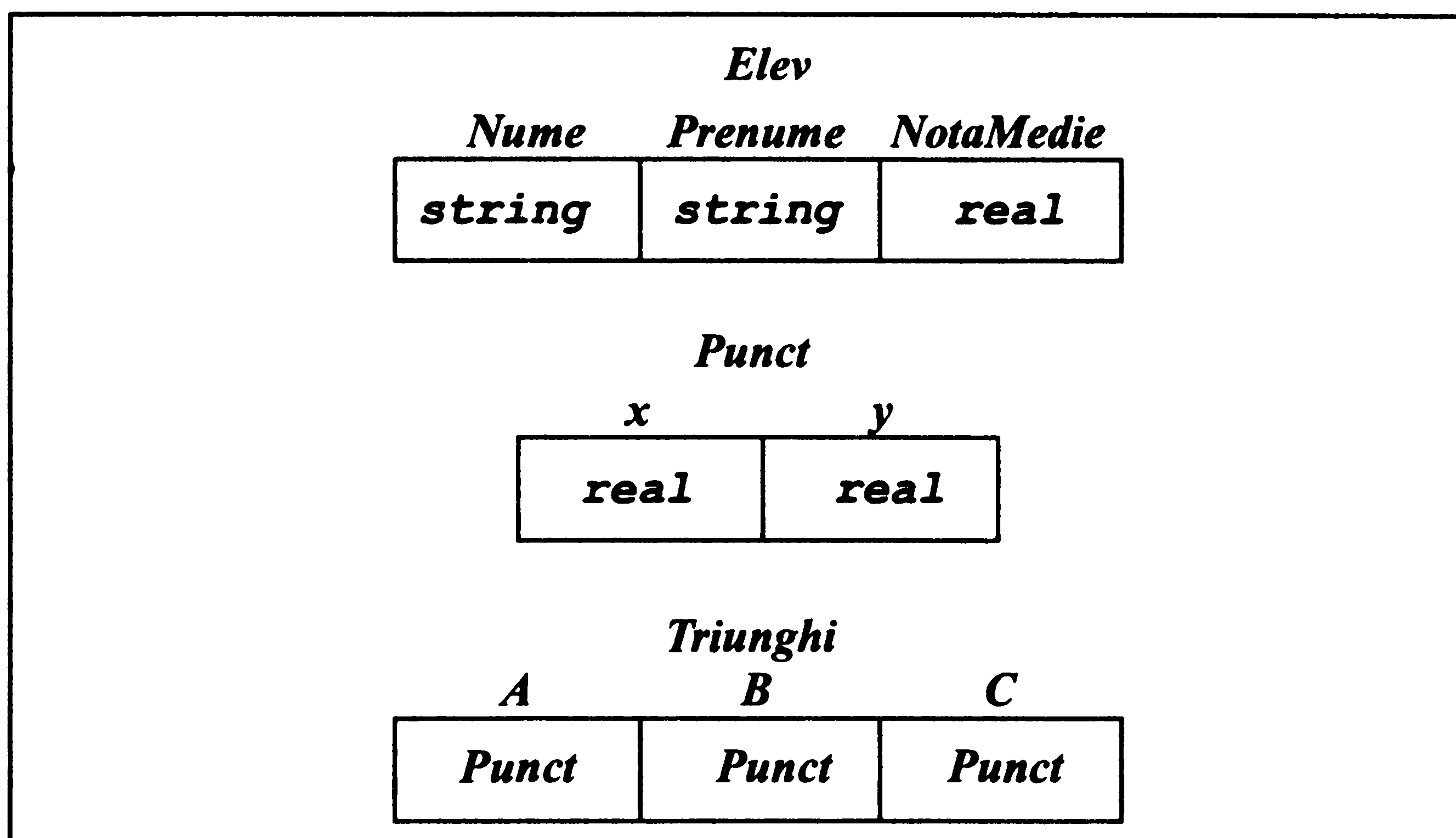
var T1, T2, T3 : Triunghi;
```

Structura datelor din exemplele în studiu este prezentată în *fig. 4.4*.

Fiind date două variabile de tip *articol* de același tip, numele variabilelor pot apărea într-o instrucțiune de atribuire. Această atribuire înseamnă copierea tuturor cîmpurilor din membrul drept în membrul stîng. De exemplu, pentru tipurile de date și variabile declarate mai sus instrucțiunile

```
E1:=E2;
T2:=T3;
P2:=P1
```

sînt corecte.



*Fig. 4.4.* Structura datelor de tip Elev, Punct și Triunghi

Fiecare componentă a unei variabile de tip **record** poate fi specificată explicit, prin numele variabilei și denumirile de câmpuri, separate prin puncte.

*Exemple:*

```
E1.Nume, E1.Prenume, E1.NotaMedie;
E2.Nume, E2.Prenume, E2.NotaMedie;
P1.x, P1.y, P2.x, P2.y;
T1.A, T1.B, T1.C, T2.A, T2.B, T2.C,
T1.A.x, T1.A.y, T2.B.x, T2.B.y;
```

Evident, componenta E1.Nume este de tip **string**; componenta P1.x este de tip **real**; componenta T1.A este de tip **Punct**; componenta T1.A.x este de tip **real** ș.a.m.d.

Asupra componentelor datelor de tip *articol* se pot efectua toate operațiile admise de tipul câmpului respectiv. Programul ce urmează compară notele medii a doi elevi și afișează pe ecran numele și prenumele elevului cu nota medie mai bună. Se consideră că elevii au note medii diferite.

```
Program P84;
{ Date de tipul Elev }
type Elev=record
 Nume : string;
 Prenume : string;
 NotaMedie : real
end;
var E1, E2, E3 : Elev;
begin
```

```

writeln('Dați datele primului elev:');
write('Numele: '); readln(E1.Nume);
write('Prenumele: '); readln(E1.Prenume);
write('Nota medie: '); readln(E1.NotaMedie);

writeln('Dați datele elevului al doilea: ');
write('Numele: '); readln(E2.Nume);
write('Prenumele: '); readln(E2.Prenume);
write('Nota medie: '); readln(E2.NotaMedie);

 if E1.NotaMedie > E2.NotaMedie
 then E3:=E1 else E3:=E2;

writeln('Elevul cu media mai bună:');
writeln(E3.Nume, ' ', E3.Prenume,
 E3.NotaMedie : 5:2);
 readln;
end.

```

Orice tip de date **record** poate servi ca tip de bază pentru formarea altor tipuri structurate.

*Exemplu:*

```

type ListaElevilor= array[1..40] of Elev;
var LE: ListaElevilor;

```

Evident, notația LE [ i] specifică elevul i din listă; notația LE[ i] .Nume specifică numele acestui elev ș.a.m.d. Programul ce urmează citește de la tastatură datele referitoare la n elevi și afișează pe ecran numele, prenumele și nota medie a celui mai bun elev. Se consideră că elevii au note medii diferite.

```

Program P85;
{ Tablou cu componente de tipul Elev }
type Elev = record
 Nume : string;
 Prenume : string;
 NotaMedie : real
end;
 ListaElev = array[1..40] of Elev;
Var E : Elev;
 LE : ListaElev;
 n : 1..40;
 i : integer;
begin
 write('n='); readln(n);
 for i:=1 to n do
 begin
 writeln('Dați datele elevului ', i);

```



```

 write('Numele: '); readln(LE[i] .Nume);
 write('Prenumele: ');
 readln(LE[i] .Prenume);
 write('Nota Medie: ');
 readln(LE[i] .NotaMedie);
 end;
E.NotaMedie:=0;
for i:=1 to n do
 if LE[i] .NotaMedie > E.NotaMedie
 then E:=LE[i] ;
writeln('Cel mai bun elev:');
writeln(E.Nume, ' ', E.Prenume, E.NotaMedie : 5:2);
readln;
end.

```

În general, un tip de date *articol* se definește cu ajutorul diagrame-  
lor sintactice din *fig. 4.5*. În completare la articolele cu un număr fix de  
câmpuri, limbajul PASCAL permite utilizarea articolelor cu variante.  
Aceste tipuri de date se studiază în cursurile avansate de informatică.

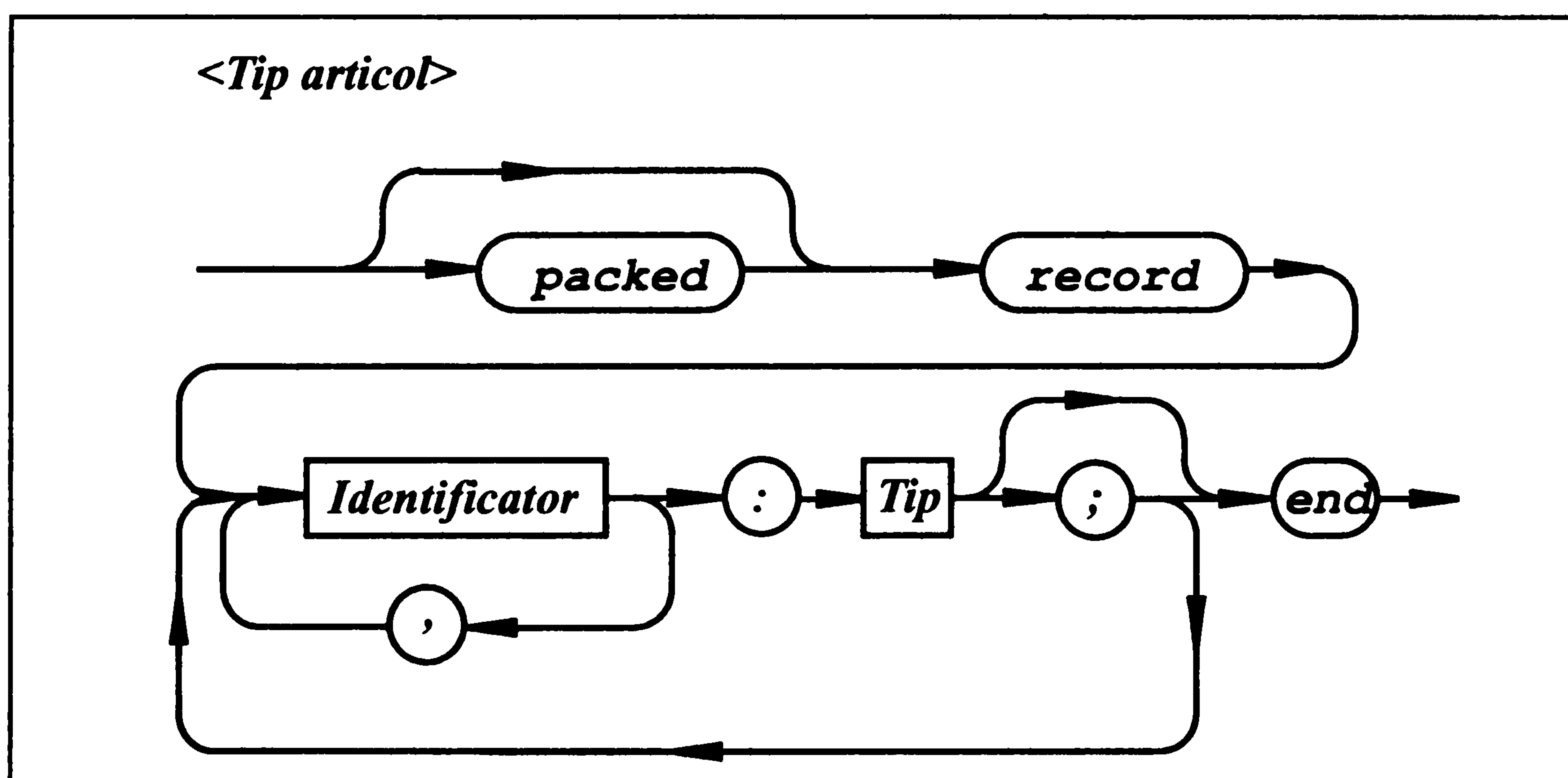


Fig. 4.5. Diagrama sintactică <Tip articol>

## Întrebări și exerciții

- ❶ Care este mulțimea de valori ale unui tip de date *articol*?
- ❷ Indicați pe diagrama sintactică din *fig. 4.5* drumurile care corespund definițiilor tipurilor de date *articol* din programele P84 și P85.
- ❸ Scrieți formulele metalingvistice pentru diagrama sintactică din *fig. 4.5*.
- ❹ Se consideră următoarele tipuri de date

```

type Data= record
 Ziua : 1..31;
 Luna : 1..12;

```

```

 Anul : integer;
 end;
 Persoana=record
 NumePrenume : string;
 DataNasterii : Data
 end;
 ListaPersoane=array [1..50] of Persoana;

```

Elaborați un program care citește de pe tastatură datele referitoare la  $n$  persoane ( $n \leq 50$ ) și afișează pe ecran:

- persoanele născute în ziua  $z$  a lunii;
- persoanele născute în luna  $l$  a anului;
- persoanele născute în anul  $a$ ;
- persoanele născute pe data  $z.l.a$ ;
- persoana cea mai în vârstă;
- persoana cea mai tânără;
- vârsta fiecărei persoane în ani, luni, zile;
- lista persoanelor care au mai mult de  $v$  ani;
- lista persoanelor în ordine alfabetică;
- lista persoanelor ordonată conform datei nașterii;
- lista persoanelor de aceeași vârstă (născuți în același an).

- ⑤ Se consideră  $n$  puncte ( $n \leq 30$ ) pe un plan euclidian. Fiecare punct  $i$  este definit prin coordonatele sale  $x_i, y_i$ . Distanța dintre punctele  $i$  și  $j$  se calculează după formula

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

Elaborați un program care afișează pe ecran punctele distanța dintre care este maximă.

- ⑥ Aria triunghiului este dată de formula lui Heron

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

unde  $p$  este semiperimetrul, iar  $a, b$  și  $c$  sînt lungimile laturilor respective. Utilizînd tipurile de date **Punct** și **Triunghi** din paragraful în studiu, elaborați un program care citește de la tastatură informațiile referitoare la  $n$  triunghiuri ( $n \leq 10$ ) și afișează pe ecran:

- aria fiecărui triunghi;
- coordoanatele vîrfurilor triunghiului cu aria maximă;
- coordoanatele vîrfurilor triunghiului cu aria minimă;
- informațiile referitoare la fiecare triunghi în ordinea creșterii ariilor.

## 4.4. INSTRUCȚIUNEA WITH

Componentele unei variabile de tip articol se specifică explicit prin numele variabilei și denumirile de cîmpuri, separate prin puncte.

De exemplu, în prezența declarațiilor



```

type Angajat=record
 NumePrenume : string;
 ZileLucrate : 1..31;
 PlataPeZi : real;
 PlataPeLuna : real;
 end;
var A : angajat;

```

componentele variabilei A se specifică prin A.NumePrenume, A.ZileLucrate, A.PlataPeZi și A.PlataPeLuna.

Întrucât numele A al variabilei de tip articol se repetă de mai multe ori, acest mod de referire a componentelor este în anumite situații greoi. Repetările obositoare pot fi evitate cu ajutorul instrucțiunii **with** (cu).

Sintaxa instrucțiunii în studiu este:

*<Instrucțiune with> ::= with <Variabilă> {, <Variabilă> }  
do <Instrucțiune>*

Diagrama sintactică este prezentată în fig. 4.6.

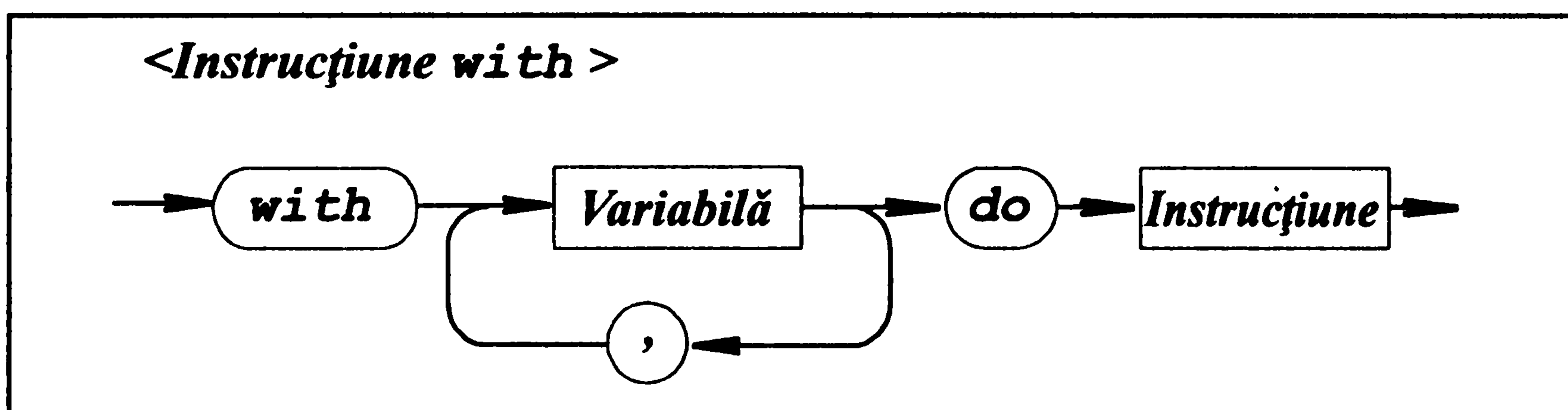


Fig. 4.6. Diagrama sintactică a instrucțiunii with

În interiorul unei instrucțiuni **with** componentele uneia sau a mai multor variabile de tip *articol* pot fi referite folosind numai numele câmpurilor respective.

*Exemplu:*

```
with A do PlataPeLuna:=PlataPeZi*ZileLucrate
```

Această instrucțiune este echivalentă cu următoarea:

```
A.PlataPeLuna:=A.PlataPeZi*A.ZileLucrate
```

Utilizarea instrucțiunii **with** necesită o atenție sporită din partea programatorului, care este obligat să specifice univoc componentele variabilelor de tip *articol*. În interiorul unei instrucțiuni **with**, la întâlnirea unui identificator, prima dată se testează dacă el poate fi interpretat ca un nume de câmp al articolului respectiv. Dacă da, identificatorul va fi interpretat ca atare, chiar dacă în acel moment este accesibilă și o variabilă avînd același nume.



*Exemplu:*

```
type Punct=record
 x : real;
 y : real;
 end;
 Segment=record
 A : Punct;
 B : Punct;
 end;
var P : Punct;
 S : Segment;
 x : integer;
```

În cazul nostru identificatorul  $x$  poate să reprezinte fie variabila de tip `integer`  $x$ , fie câmpul  $P.x$  al articolului  $P$ .

În instrucțiunea

```
x:=1
```

identificatorul  $x$  se referă la variabila de tip `integer`  $x$ .

În instrucțiunea

```
with P do x:=1
```

identificatorul  $x$  se referă la câmpul  $P.x$  al variabilei de tip articol  $P$ .

Întrucât variabila de tip articol  $S$  nu conține nici un câmp cu numele  $S.x$ , în instrucțiunea

```
with S do x:=1
```

identificatorul  $x$  va fi interpretat ca variabila de tip `integer`  $x$ .

O instrucțiune de forma

```
with v_1, v_2, \dots, v_n do <Instrucțiune> ,
```

unde  $v_1, v_2, \dots, v_n$  sînt variabile de tip *articol*, este echivalentă cu instrucțiunea

```
with v_1 do
with v_2 do
 { ... }
with v_n do <Instrucțiune> .
```

Evident, componentele variabilelor  $v_1, v_2, \dots, v_n$  trebuie specificate univoc prin denumirile câmpurilor respective.

De exemplu, pentru variabilele  $P$  și  $S$ , declarate mai sus, putem scrie:

```
with P, S do
begin
 x:=1.0; { referire la P.x }
 y:=1.0;
 A.x:=0; { referire la S.A.x }
```

```

 A.y:=0;
 B.x:=2.0; { referire la S.B.x}
 B.y:=2.0;
end;

```

În mod obișnuit, instrucțiunea **with** se utilizează numai în cazurile în care se ajunge la o reducere semnificativă a textului unui program.

## Întrebări și exerciții

- ❶ Indicați pe diagrama sintactică din *fig. 4.6* drumurile care corespund instrucțiunilor **with** din exemplele paragrafului în studiu.
- ❷ Care este destinația instrucțiunii **with**?
- ❸ Utilizând instrucțiunea **with**, excludeți din programele P84 și P85 din paragraful precedent repetările de genul

```

E1.Nume, E1.Prenume, ...,
LE[i].Nume, LE[i].Prenume.

```

- ❹ Se consideră următoarele tipuri de date:

```

Type Angajat=record
 NumePrenume : string;
 ZileLucrate : 1..31;
 PlataPeZi : real;
 PlataPeLuna : real;

```

**end;**

```

ListaDePlata=array[1..50] of Angajat;

```

Plata pe lună a fiecărui angajat se calculează înmulțind plata pe zi cu numărul de zile lucrate. Elaborați un program care:

- a) calculează plata pe lună a fiecărui angajat;
  - b) calculează salariul mediu al angajaților incluși în listă;
  - c) afișează pe ecran datele despre angajații cu plata lunară maximă;
  - d) afișează lista angajaților ordonată alfabetic;
  - e) afișează lista angajaților în ordinea creșterii plăților pe zi;
  - f) ordonează lista angajaților în ordinea creșterii plăților pe lună;
  - g) afișează lista angajaților în ordinea creșterii numărului de zile lucrate.
- ❺ Un cerc poate fi definit prin coordonatele  $x$ ,  $y$  și raza  $r$ . Elaborați un program care citește de la tastatură datele referitoare la  $n$  cercuri ( $n \leq 50$ ) și afișează pe ecran:
    - a) coordonatele centrului și raza cercului cu aria maximă;
    - b) numărul de cercuri incluse în cercul cu raza maximă și coordonatele centrelor respective;
    - c) coordonatele centrului și raza cercului cu aria minimă;
    - d) numărul de cercuri în care este inclus cercul cu raza minimă și coordonatele centrelor respective.

## 4.5. TIPURI DE DATE *MULȚIME* (SET)

Un tip de date *mulțime* (**set**) se definește în raport cu un tip de bază care trebuie să fie ordinal:

*<Tip mulțime>* ::= [**packed**] **set of** *<Tip>*

Valorile unui tip de date **set** sînt mulțimi formate din valorile tipului de bază. Dacă tipul de bază are  $n$  valori, tipul mulțime va avea  $2^n$  valori. În implementările limbajului valoarea lui  $n$  este limitată, de regulă  $n \leq 256$ .

În PASCAL o mulțime poate fi specificată enumerîndu-i-se elementele între parantezele pătrate “[ ” și “] ”, care țin locul acoladelor din matematică.

Notăția [ ] reprezintă mulțimea vidă.

*Exemple:*

```
type Indice=1..10;
 Zi=(L, Ma, Mi, J, V, S, D);
 MultimeIndicii=set of Indice;
 ZileDePrezenta=set of Zi;
var MI : MultimeIndicii;
 ZP : ZileDePrezenta;
```

Tipul ordinal *Indice* are  $n = 10$  valori: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Tipul *MultimeIndicii* are  $2^{10} = 1024$  de valori, și anume:

[ ], [ 1 ], [ 2 ], ..., [ 1, 2 ], [ 1, 3 ], ...,  
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ].

Prin urmare, variabila MI poate să aibă oricare din aceste valori, de exemplu:

MI := [ 1, 3 ].

Tipul ordinal *Zi* are  $n = 7$  valori: L, Ma, Mi, J, V, S, D. Tipul *ZileDePrezenta* are  $2^7 = 128$  de valori, și anume

[ ], [ L ], [ Ma ], [ Mi ], ..., [ L, Ma ], [ L, Mi ], ...,  
[ L, Ma, Mi, J, V, S, D ].

Variabila ZP poate să aibă oricare dintre aceste valori, de exemplu

ZP := [ L, Ma, Mi, V ].

O valoare de tip *mulțime* poate fi specificată printr-un **constructor** (generator) de mulțime. Diagrama sintactică a unității gramaticale *<Constructor mulțime>* este prezentată în *fig. 4.7*. Un constructor conține specificarea elementelor mulțimii, separate prin virgule, și incluse între paranteze pătrate. Un element poate să fie o valoare concretă a tipului de bază sau un interval de forma:



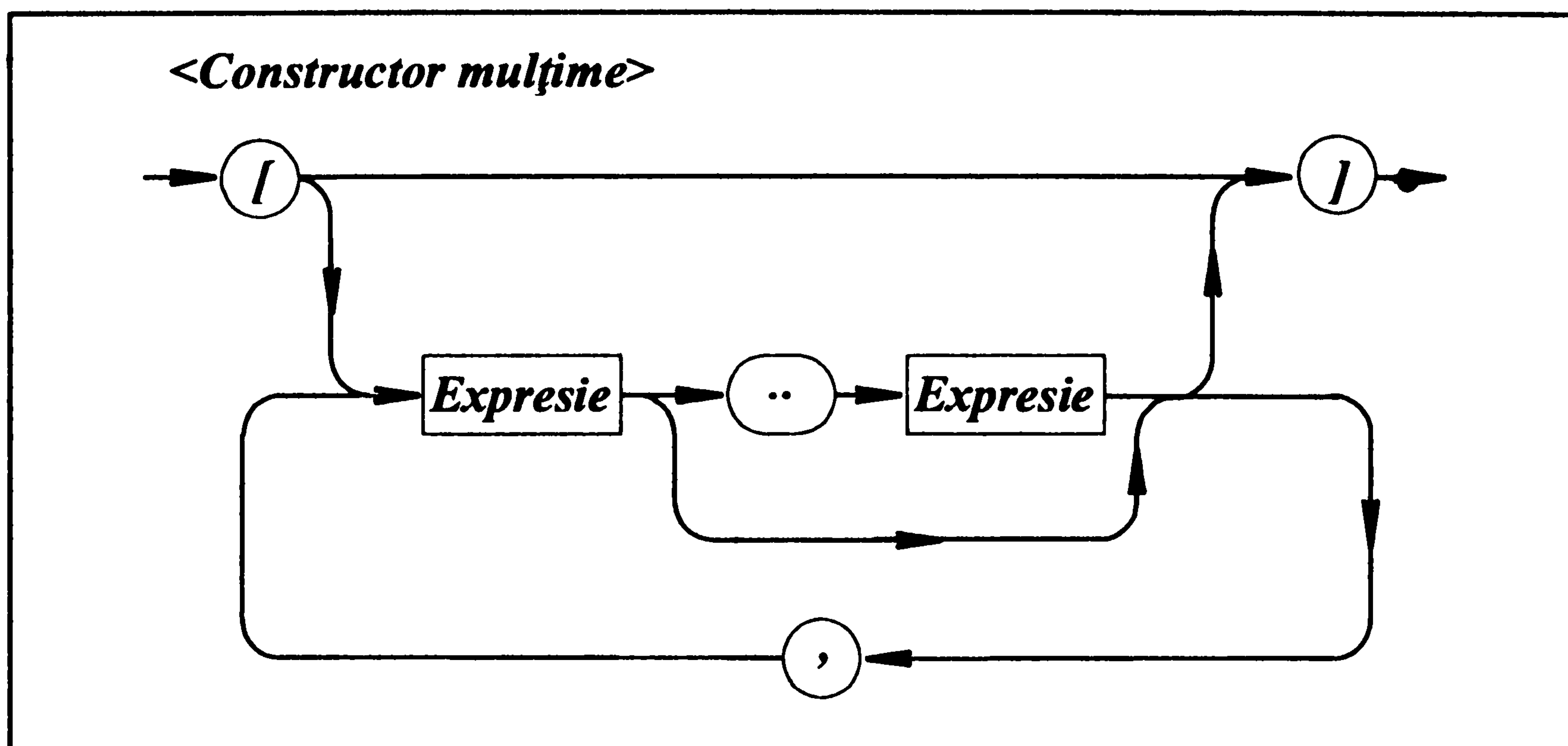


Fig. 4.7. Diagrama sintactică <Constructor mulțime>

**<Expresie> .. <Expresie>**

Valorile expresiilor în studiu precizează limitele inferioare și superioare ale intervalului.

*Exemple :*

[ ]  
 [ 1, 2, 3, 8]  
 [ 1..4, 8..10]  
 [ i-k..i+k]  
 [ L, Ma, V..D]

Asupra valorilor unui tip de date *mulțime* se pot efectua operațiile uzuale:

- + reuniunea;
- \* intersecția;
- diferența,

rezultatul fiind de tip *mulțime* și operațiile relaționale:

- = egalitatea;
- <> inegalitatea;
- <=, => incluziunea;
- in** apartenența,

rezultatul fiind de tip boolean.

Programul ce urmează afișează pe ecran rezultatele operațiilor +, \* și -, efectuate asupra valorilor de tip MultimeIndicii.

```
Program P86;
 { Date de tip MultimeIndicii }
type Indice=1..10;
```

```

 MultimeIndicii=set of Indice;
var A, B, C : MultimeIndicii;
 i : integer;
begin
 A:=[1..5, 8]; { A conține 1,2,3,4,5,8 }
 B:=[1..3, 9, 10]; { B conține 1,2,3,9,10 }
 C:=[]; { C este o mulțime vidă }

 C:=A+B; { C conține 1,2,3,4,5,8,9,10 }
 writeln('Reuniune');
 for i:=1 to 10 do
 if i in C then write(i:3);
 writeln;

 C:=A*B; { C conține 1, 2, 3 }
 writeln('Intersecție');
 for i:=1 to 10 do
 if i in C then write(i:3);
 writeln;

 C:=A-B; { C conține 4, 5, 8 }
 writeln('Diferența');
 for i:=1 to 10 do
 if i in C then write(i:3);
 writeln;
 readln;
end.

```

Spre deosebire de tablouri și articole, componentele cărora pot fi referite direct, respectiv prin indicii și denumiri de câmpuri, elementele unei mulțimi nu pot fi referite. Se admite numai verificarea apartenenței elementului la o mulțime (operația relațională **in**). În pofida acestui fapt, utilizarea tipurilor de date *mulțime* mărește viteza de execuție și îmbunătățește lizibilitatea programelor PASCAL. De exemplu, instrucțiunea:

```

if (c='A') or (c='E') or (c='I')
 or (c='O') or (c='U') then ...

```

poate fi înlocuită cu o instrucțiune mai simplă:

```

if c in ['A', 'E', 'I', 'O', 'U'] then ...

```

Un alt exemplu sugestiv este utilizarea tipurilor de date *mulțime* în calcularea numerelor prime mai mici decât un număr natural dat  $n$ . Pentru aceasta se folosește algoritmul *Ciurul (sita) lui Eratostene*:

- 1) în sită se depun numerele 2, 3, 4, ...,  $n$ ;
- 2) din sită se extrage cel mai mic număr  $i$ ;
- 3) numărul extras se include în mulțimea numerelor prime;
- 4) din sită se elimină toți multiplii  $m$  ai numărului  $i$ ;
- 5) procesul se încheie când sita s-a golit .

```

Program P87;
 { Ciurul (sita) lui Eratostene }
 const n=50;
 type MultimeDeNumere=set of 1..n;
 var Sita, NumerePrime : MultimeDeNumere;
 i, m : integer;

 begin
 { 1} Sita:=[2..n] ;
 NumerePrime:=[] ;
 i:=2;
 repeat
 { 2} while not (i in Sita) do i:=succ(i);
 { 3} NumerePrime:=NumerePrime+[i] ;
 write(i:4);
 m:=i;
 { 4} while m<=n do
 begin Sita:=Sita-[m] ; m:=m+i; end;
 { 5} until Sita=[] ;
 writeln;
 readln;
 end.

```

Correspondența dintre punctele algoritmului și instrucțiunile care le exprimă este indicată în comentariile din partea stîngă a liniilor de program.

### Întrebări și exerciții

❶ Enumerați valorile posibile ale variabilelor din declarațiile ce urmează:

```

var V : set of 'A'..'C';
 S : set of (A, B, C);
 I : set of '1'..'2';
 J : set of 1..2;

```

❷ Comentați următorul program :

```

Program P88;
 { Eroare }
 type Multime=set of integer;
 var M : Multime;
 i : integer;

 begin
 M:=[1, 8, 13] ;
 for i:=1 to MaxInt do
 if i in M then writeln(i);
 end.

```

❸ Se consideră următoarele declarații:



```

type Culoare=(Galben,Verde,Albastru,Violet);
 Nuanta = set of Culoare;
var NT : Nuanta;

```

Care sînt valorile posibile ale variabilei NT?

- ④ Scrieți formula metalingvistică care corespunde diagramei sintactice <Constructor mulțime> din fig. 4.7.
- ⑤ Se consideră tipul de date MultimeIndicii din paragraful în studiu. Precizați mulțimile specificate de constructorii ce urmează:
 

|                      |                       |
|----------------------|-----------------------|
| a) [ ] ;             | f) [ 4..3] ;          |
| b) [ 1..10] ;        | g) [ 1..3, 7..6, 9] ; |
| c) [ 1..3, 9..10] ;  | h) [ 4-2..7+1] ;      |
| d) [ 1+1, 4..7, 9] ; | i) [ 7-5..4+4] ;      |
| e) [ 3, 7..9] ;      | j) [ 6, 9, 1..2] .    |
- ⑥ Elaborați un program care afișează pe ecran toate submulțimile mulțimii {1, 2, 3, 4}.
- ⑦ Elaborați un program care afișează pe ecran toate submulțimile mulțimii {'A', 'B', 'C', 'D'}.
- ⑧ Se dă un șir de caractere în care cuvintele sînt separate fie prin spațiu, fie prin caracterele *punct*, *virgulă*, *punct și virgulă*, *semnul exclamării* și *semnul întrebării*. Elaborați un program care afișează pe ecran cuvintele șirului de caractere citit de la tastatură.
- ⑨ Se dă un șir de caractere. Elaborați un program care afișează pe ecran numărul de vocale din șir.
- ⑩ Elaborați un program care citește de la tastatură două șiruri de caractere și afișează pe ecran :
  - a) caracterele care se întîlnesc cel puțin în unul din șiruri;
  - b) caracterele care apar în ambele șiruri;
  - c) caracterele care apar în primul și nu apar în șirul al doilea.
- ⑪ Să se scrie un program care verifică dacă numele unei persoane este introdus corect (numele este un șir de caractere ce nu conține cifre).
- ⑫ În implementările actuale ale limbajului numărul valorilor tipului de bază al unui tip *mulțime* este limitat , obișnuit  $n \leq 256$ . În consecință, programul P87 nu poate calcula numere prime mai mari decît  $n$ . Elaborați un program pentru calcularea numerelor prime din intervalul 8, ..., 10 000.  
*Indicație:* Sita din algoritmul lui Eratostene poate fi reprezentată printr-un tablou, componentele căruia sînt mulțimi.

## 4.6. GENERALITĂȚI DESPRE FIȘIERE

Prin **fișier** se înțelege o structură de date care constă dintr-o secvență de componente. Fiecare componentă din secvență are același tip, denumit *tip de bază*. Numărul componentelor din secvență nu este fixat, însă sfîrșitul secvenței este indicat de un simbol special, notat *EOF* (*End of File* — sfîrșit de fișier). Fișierul care nu conține nici o componentă se numește *fișier vid*.

Un tip de date fișier se definește printr-o declarație de forma:

$\langle \text{Tip fişier} \rangle ::= [\text{packed}] \text{ file of } \langle \text{Tip} \rangle ;$

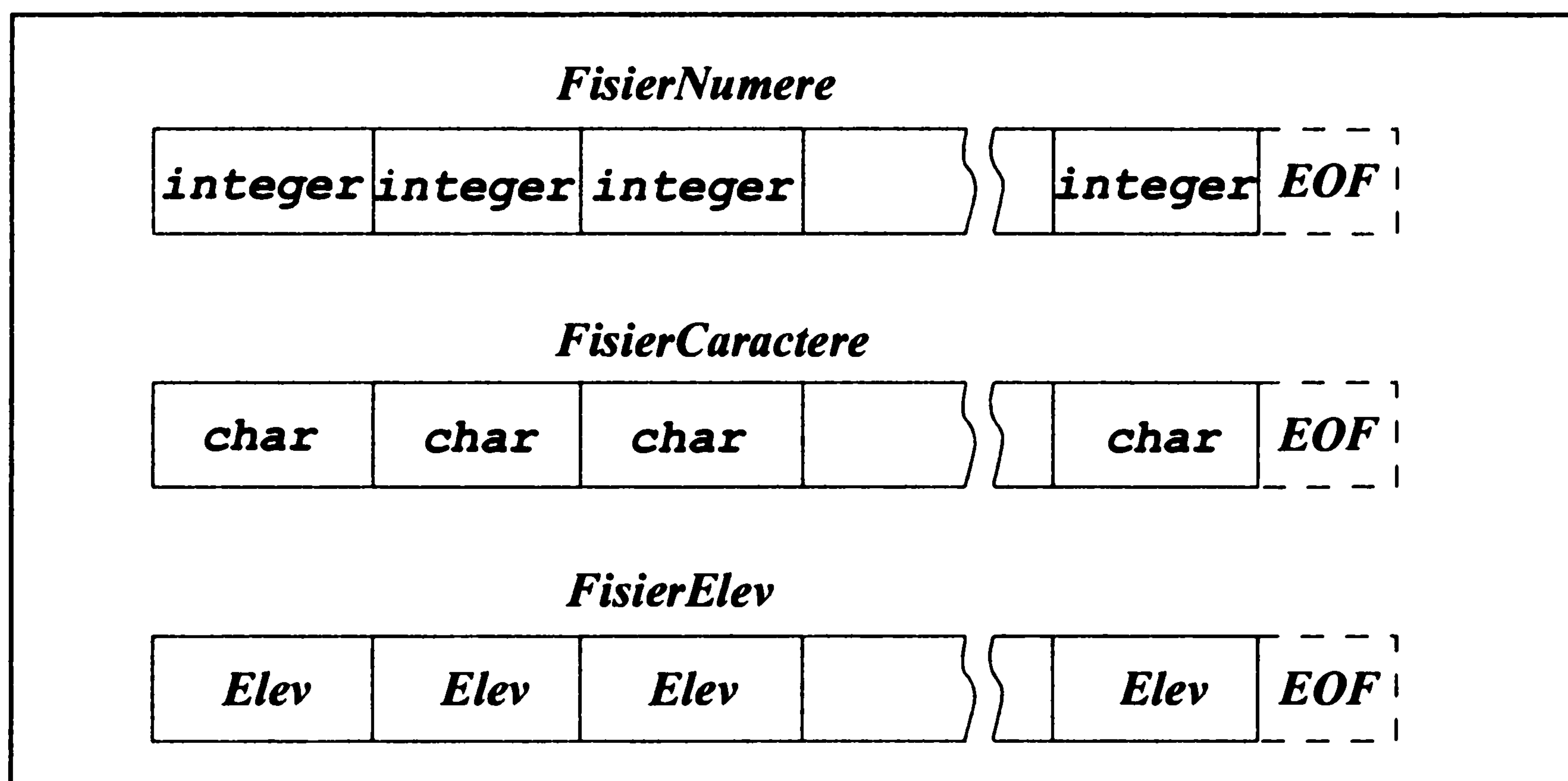
unde  $\langle \text{Tip} \rangle$  este tipul de bază. Tipul de bază este un tip arbitrar, exceptînd tipul *fişier* (nu există “fişier de fişiere”).

*Exemple:*

```

1) type FisierNumere=file of integer;
 var FN : FisierNumere;
 n : integer;
2) type FisierCaractere=file of char;
 var FC : FisierCaractere;
 c : char;
3) type Elev=record
 Nume : string;
 Prenume: string;
 NotaMedie : real;
 end;
 FisierElevi=file of Elev;
var FE : FisierElevi
 E : Elev;
```

Structura datelor în studiu este prezentată în *fig. 4.8*. Subliniem faptul că elementul *EOF*, care indică sfîrşitul secvenţei nu este o componentă a fişierului.



*Fig. 4.8.* Structura datelor de tip *FisierNume*, *FisierCaractere* şi *FisierElevi*

Variabilele FN, FC, FE ş.a. de tip *fişier* se numesc **fişiere PASCAL** sau, pur şi simplu, *fişiere*. Spre deosebire de celelalte tipuri de date, valorile cărora se păstrează în memoria internă a calculatorului, datele fişierelor PASCAL se păstrează pe suporturile de informaţie ale echipamentelor periferice (discuri şi benzi magnetice, discuri optice, hîrtia imprimantei sau dispozitivului de citit documente ş. a.). Informaţia pe suporturile în studiu este organizată în formă de **fişiere ex-**



terne în conformitate cu cerințele sistemului de operare. Prin urmare, înainte de a fi utilizată, o variabilă fișier trebuie asociată cu un fișier extern. Metodele de asociere sînt specifice implementării limbajului și sistemului de operare al calculatorului-gazdă.

În limbajul-standard asocierea se realizează prin includerea variabilelor de tip *fișier* ca argumente în antetul de program.

În Turbo PASCAL asocierea unei variabile de tip *fișier*  $f$  cu un fișier extern se realizează prin apelul de procedură

```
assign(f , s);
```

unde  $s$  este o expresie de tip **string** care specifică fișierul extern.

*Exemple:*

```
assign(FN, 'A:\REZULTAT\R.DAT')
```

– fișierul FN se asociază cu fișierul extern R.DAT din directorul REZULTAT de pe discul A;

```
assign(FC, 'C:\A.CHR')
```

- fișierul FC se asociază cu fișierul A.CHR din directorul rădăcină al discului C;

```
write('Dați un nume de fișier:');
```

```
readln(str);
```

```
assign(FE, str)
```

- fișierul FE se asociază cu un fișier extern numele căruia este citit de la tastatură și depus în variabila de tip **string** str.

După executarea instrucțiunii `assign( $f$ ,  $s$ )` toate operațiile referitoare la fișierul PASCAL  $f$  se vor efectua asupra fișierului extern  $s$ .

Cele mai uzuale operații asupra unui fișier sînt citirea și scrierea unei componente.

**Citirea** unei componente se realizează printr-un apel de forma

```
read(f , v),
```

unde  $v$  este o variabilă declarată cu tipul de bază al fișierului  $f$ .

**Scrierea** unei componente se realizează printr-un apel de forma:

```
write(f , e),
```

unde  $e$  este o expresie asociată cu tipul de bază al fișierului  $f$ .

*Exemple:*

```
read(FN, n);
```

```
write(FC, c);
```

```
read(FE, E);
```

După **tipul operațiilor permise** asupra componentelor, fișierele se clasifică în:

– fișiere de intrare (este permisă numai citirea);



- fișiere de ieșire (este permisă numai scrierea);
- fișiere de actualizare (sînt permise scrierea și citirea).

După **modul de acces** la componente, fișierele se clasifică în:

- fișiere cu acces secvențial sau secvențiale (accesul la componenta  $i$  este permis după ce s-a citit/scriș componenta  $i - 1$ );
- fișiere cu acces aleator sau direct (orice componentă se poate referi direct prin numărul ei de ordine  $i$  în fișier).

Menționăm că în limbajul-standard sînt permise numai fișiere secvențiale de intrare sau ieșire.

Tipul fișierului (de intrare, ieșire sau de actualizare) și modul de acces (secvențial sau direct) se stabilesc printr-o operație de validare, numită **deschidere a fișierului**. Pentru aceasta, în limbajul-standard se utilizează următoarele proceduri:

`reset (f)` — pregătește un fișier existent pentru citire;

`rewrite (f)` — creează un fișier vid și îl pregătește pentru scriere.

Cînd prelucrarea componentelor se termină, fișierul trebuie închis. La **închiderea** unui fișier sistemul de operare înscrie, dacă e necesar, elementul *EOF*; înregistrează fișierul extern nou creat în directorul respectiv ș.a.m.d.

În limbajul-standard se consideră că fișierele vor fi închise implicit la terminarea execuției programului respectiv. În Turbo PASCAL fișierul  $f$  ce închide prin apelul de procedură `close (f)`.

În concluzie, prezentăm ordinea în care trebuie apelate procedurile destinate prelucrării datelor de tip *fișier*:

1) `assign (f, s)` — asocierea *fișierului* PASCAL  $f$  cu fișierul extern  $s$ ;

2) `reset (f) / rewrite (f)` — deschiderea fișierului  $f$  pentru citire/scriere;

3) `read (f, v) / write (f, e)` — citirea/scrierea unei componente a fișierului  $f$ ;

4) `close (f)` — închiderea fișierului  $f$ .

După închiderea fișierului, variabila  $f$  poate fi asociată cu un alt fișier extern.

Întrucît valorile variabilelor de tip fișier se păstrează pe suporturile externe de informație, în PASCAL **atribuirile de fișiere sînt interzise**.

## Întrebări și exerciții

- ❶ Explicați termenii *fișier PASCAL*, *fișier extern*.
- ❷ Unde se păstrează datele unui fișier PASCAL? Care este destinația procedurii `assign`?
- ❸ Reprezentați pe un desen structura următoarelor tipuri de date:

```

a) type Tabel=array[1..5, 1..10] of real;
 FisierTabele =file of Tabel;
b) type Multime=set of 'A'..'C';
 FisierMultimi=file of Multime;
c) type Punct=record x, y:real end;
 Segment=record A, B:Punct end;
 FisierSegmente=file of Segment;

```

- ④ Pentru ce sînt necesare operațiile de deschidere și închidere a fișierelor? Cum se realizează aceste operații?
- ⑤ Explicați destinația procedurilor read și write. Ce tip trebuie să aibă variabila  $v$  într-un apel de forma read( $f, v$ )? Ce tip trebuie să aibă expresia  $e$  într-un apel de forma write( $f, e$ )?
- ⑥ Cum se clasifică fișierele în funcție de operațiile permise și modul de acces?
- ⑦ Variabilele A și B sînt introduse prin declarația

```
var A, B : file of integer;
```

Este oare corectă instrucțiunea?

```
A:=B
```

Argumentați răspunsul.

## 4.7. FIȘIERE SECVENȚIALE

Fie definițiile PASCAL

```
type FT=file of T;
var f : FT; v : T;
```

prin care au fost introduse tipul *fișier* FT cu tipul de bază T; variabila de tip *fișier*  $f$  și variabila  $v$  de tipul T.

Pentru a deschide un **fișier secvențial de ieșire** se apelează procedura rewrite( $f$ ). În continuare în fișier se înscriu componentele respective. O componentă se înscrie printr-un apel de forma

```
write(f, e),
```

unde  $e$  este o expresie de tipul T. O instrucțiune de forma

```
write(f, e_1, e_2, \dots, e_n)
```

este echivalentă cu secvența de instrucțiuni

```
write(f, e_1); write(f, e_2); ...; write(f, e_n).
```

După înscrierea ultimei componente fișierul trebuie închis.

*Exemplu:*

```
Program P89;
{ Crearea unui fișier cu componente de tipul Elev }
type Elev=record
```



```

 Nume : string;
 Prenume : string;
 NotaMedie : real
 end;
 FisierElevi=file of Elev;
var FE : FisierElevi;
 E : Elev;
 str : string;
 i, n : integer;
begin
write('Dați numele fișierului de creat: ');
readln(str);

assign(FE, str);
 { asociază FE cu numele din str }
rewrite(FE);
 { deschide FE pentru scriere }

write('Dați numărul de elevi: '); readln(n);
for i:=1 to n do
 begin
 writeln('Dați datele elevului ', i);
 { citește cîmpurile variabilei E
 de la tastatură }
 write('Numele: '); readln(E.Nume);
 write('Prenumele: '); readln(E.Prenume);
 write('Nota medie: '); readln(E.NotaMedie);
 write(FE, E); { scrie E în FE }
 end;
close(FE); { închide FE }
readln;
end.

```

Pentru a **deschide un fișier secvențial de intrare** se apelează procedura *reset (f)*. Componenta curentă se citește din fișier printr-un apel de forma:

*read (f, v).*

O instrucțiune de forma

*read (f, v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>n</sub>)*

este echivalentă cu secvența de instrucțiuni

*read (f, v<sub>1</sub>); read (f, v<sub>2</sub>); ..., read (f, v<sub>n</sub>).*

Sfârșitul de fișier este semnalizat de funcția booleană *eof (f)* care ia valoarea *true* după citirea ultimei componente.

*Exemplu:*



```

Program P90;
 { Citirea unui fișier cu componente
 de tipul Elev }
 type Elev=record
 Nume : string;
 Prenume : string;
 NotaMedie : real
 end;
 FisierElevi=file of Elev;
var FE : FisierElevi;
 E : Elev;
 str : string;
begin
 write('Dați numele fișierului de citit: ');
 readln(str);

 assign(FE, str); { asociază FE cu numele din str }
 reset(FE); { deschide FE pentru citire }
 while not eof(FE) do
 begin
 read(FE, E); { citește E din FE }
 writeln(E.Nume, ' ', E.Prenume,
 E.NotaMedie : 5:2);
 { afișează E pe ecran }

 end;
 close(FE); { închide FE }
 readln;
end.

```

Subliniem faptul că numărul de componente ale unui fișier nu este cunoscut din declarația tipului respectiv. Teoretic, într-un fișier secvențial de ieșire pot fi înscrise un număr infinit de componente. Practic, numărul componentelor este limitat de capacitatea de memorare a suportului extern de informație. Citirea consecutivă a componentelor unui fișier secvențial de intrare se încheie când se ajunge la elementul *EOF*.

## Întrebări și exerciții

- ❶ Câte componente poate avea un fișier? În ce ordine se scriu/citesc componentele unui fișier secvențial?
- ❷ Se consideră următoarele tipuri de date:

```

type Data=record
 Ziua : 1..31;
 Luna : 1..12;
 Anul : integer
 end;
 Persoana=record
 NumePrenume : string;
 DataNașterii : Data

```

**end;**

FisierPersoane=**file of** Persoana;

Elaborați un program care citește de la tastatură datele referitoare la  $n$  persoane și le înregistrează într-un fișier.

Creați fișierele FILE1.PRS, FILE2.PRS, FILE3.PRS care trebuie să conțină datele referitoare, respectiv, la 2, 7 și 10 persoane.

③ Elaborați un program care citește fișiere create de programul din exercițiul precedent și afișează pe ecran:

- a) toate persoanele din fișier;
- b) persoanele născute în anul  $a$ ;
- c) persoanele născute pe data  $z.l.a$ ;
- d) persoana cea mai în vârstă;
- e) persoana cea mai tânără.

④ Elaborați un program care afișează pe ecran media aritmetică a numerelor înscrise într-un fișier de tipul **file of** real.

⑤ Într-un fișier de tipul **file of** char sînt înscrise caractere arbitrare. Elaborați un program care afișează pe ecran numărul vocalelor din fișier.

⑥ Comentati următorul program:

```
Program P91;
{ Eroare }
type FisierNumere=file of integer;
var FN : FisierNumere;
 i : integer;
 r : real;
 s : string;
begin
 Writeln('Dați numele fișierului de creat: ');
 readln(s);
 assign(FN, s);
 rewrite(FN);
 i:=1;
 write(FN, i);
 i:=10;
 write(FN, i);
 r:=20;
 write(FN, r);
 close(FN);
end.
```

## 4.8. FIȘIERE CU ACCES DIRECT

Acest paragraf se referă în întregime la implementarea Turbo PASCAL.

În Turbo PASCAL variabilele de tip **file of**  $T$  permit atât citiri, cât și scrieri, indiferent dacă ele au fost deschise cu **rewrite** sau



reset. Pentru aceasta fiecare componentă a fișierului are asociat un număr, numit *numărul componentei*. Prima componentă are numărul 0. Cea de-a doua componentă are numărul 1 ș.a.m.d. Numărul de componente ale unui fișier  $f$  poate fi determinat cu ajutorul funcției standard `FileSize(f)`.

În mod normal, accesul la componentele fișierului este secvențial. Aceasta înseamnă că după citirea/scrierea unei componente, poziția curentă de fișier se deplasează la următoarea componentă, în sensul ordonării numerice. Accesul direct se bazează pe procedura de căutare `Seek(f, i)`, care mută poziția curentă în fișier pe componenta  $i$ . După această poziționare componenta astfel aleasă poate fi citită/scrișă. Funcția-standard `FilePos(f, i)` permite determinarea poziției curente în fișier.

*Exemplu:*

```

Program P92;
 { Accesul direct la fișiere de tipul FisierElev }
 type Elev=record
 Nume : string;
 Prenume : string;
 NotaMedie : real
 end;
 FisierElevi=file of Elev;
var FE : FisierElevi;
 E : Elev;
 str : string;
 i, n : integer;
 r : char;
begin
 write('Dați numele fișierului: ');
 readln(str);
 assign(FE, str);
 reset(FE);
 n:=FileSize(FE);
 writeln('Fisierul', str, 'are', n, 'componente');
 writeln('Numerația componentelor: 0..', n-1);
 write('Dați numărul componentei: '); readln(i);
 if i>n-1 then
 writeln('Eroare: componentă inexistentă')
 else
 begin
 Seek(FE, i);
 { poziționare pe componenta i }
 read(FE, E);
 { citirea componentei selectate }
 writeln(E.Nume, ' ', E.Prenume,
 E.NotaMedie :5:2);
 write('Modificăm componenta? [d/n]');
 end
 end

```



```

 readln(r);
 if r='d' then
 begin
 write('Numele: '); readln(E.Nume);
 write('Prenumele: ');
 readln(E.Prenume);
 write('Nota medie: ');
 readln(E.NotaMedie);
 Seek(FE, i);
 { poziționare pe componenta i }
 write(FE, E);
 { scrierea componentei modificate }
 end;
 end;
 close(FE);
 readln;
end.

```

Amintim că fișierele de tipul `FisierElevi` pot fi create cu ajutorul programului P89 din paragraful precedent.

## Întrebări și exerciții

- ❶ Prin ce se deosebește accesul direct de cel secvențial?
- ❷ Care este destinația funcțiilor și procedurii `FileSize`, `FilePos`, `Seek`?
- ❸ Creați cu ajutorul programului P89 fișierele `CLASA0.ELV`, `CLASA1.ELV` și `CLASA5.ELV`. Fișierele trebuie să conțină, respectiv, 0, 1 și 5 componente. Verificați funcționarea programului P92 pentru fiecare din fișierele create.
- ❹ Elaborați un program care asigură accesul direct la componentele fișierelor de tipul `FisierPersoane` (vezi exercițiul 2 din paragraful precedent). Programul va afișa la ecran și va modifica, dacă se va cere, componenta indicată de utilizator.
- ❺ Se consideră următoarele tipuri de date:

```

type Angajat= record
 NumePrenume : string;
 ZileLucrate : 1..31;
 PlataPeZi : real;
 PlataPeLuna : real
end;

```

`FisierAngajati=file of Angajat;`

Elaborați un program, care:

- a) citește de la tastatură datele referitoare la fiecare angajat și creează fișiere de tipul `FisierAngajati`;
- b) modifică, dacă e necesar, câmpurile `ZileLucrate`, `PlataPeZi` și `PlataPeLuna` ale componentelor indicate din fișierul respectiv;
- c) calculează salariul mediu al angajaților incluși în fișier;
- d) afișează la ecran datele despre angajații cu plata lunară maximală;
- e) ordonează angajații din fișier în ordine alfabetică;
- f) ordonează angajații din fișier în ordinea creșterii plăților pe lună;

- g) adaugă noi angajați în fișierul existent;
- h) exclude din fișier persoanele concediate.

## 4.9. FIȘIERE TEXT

E cunoscut faptul că datele fișierelor PASCAL se păstrează pe suporturile externe de informație. În cazul fișierelor definite prin declarații de forma **file of T** componentele de tip *T* sînt reprezentate pe suporturile respective în **forma internă**, și anume, prin secvențe de cifre binare. Acest mod de reprezentare a datelor este convenabil în cazul memoriilor externe (discurile și benzile magnetice, discurile optice ș. a.). În cazul echipamentelor de intrare-ieșire (tastatura, ecranul, imprimanta etc.) datele respective trebuie reprezentate în **forma externă**, și anume, prin secvențe de caractere.

Pentru a facilita interacțiunea între om și sistemul de calcul, în PASCAL informația destinată utilizatorului se reprezintă în formă de fișiere *text*. Un fișier *text* este format dintr-o secvență de caractere divizată în linii (fig. 4.9). Lungimea liniilor este variabilă. Sfîrșitul fiecărei linii este indicat de un element special, notat *EOL* (*End Of Line* — sfîrșit de linie). Întrucît lungimea liniilor este variabilă, poziția unei linii în cadrul fișierului nu poate fi calculată din timp. În consecință, accesul la componentele fișierelor *text* este **secvențial**.

Un fișier *text* se definește printr-o declarație de forma

```
var f : text;
```

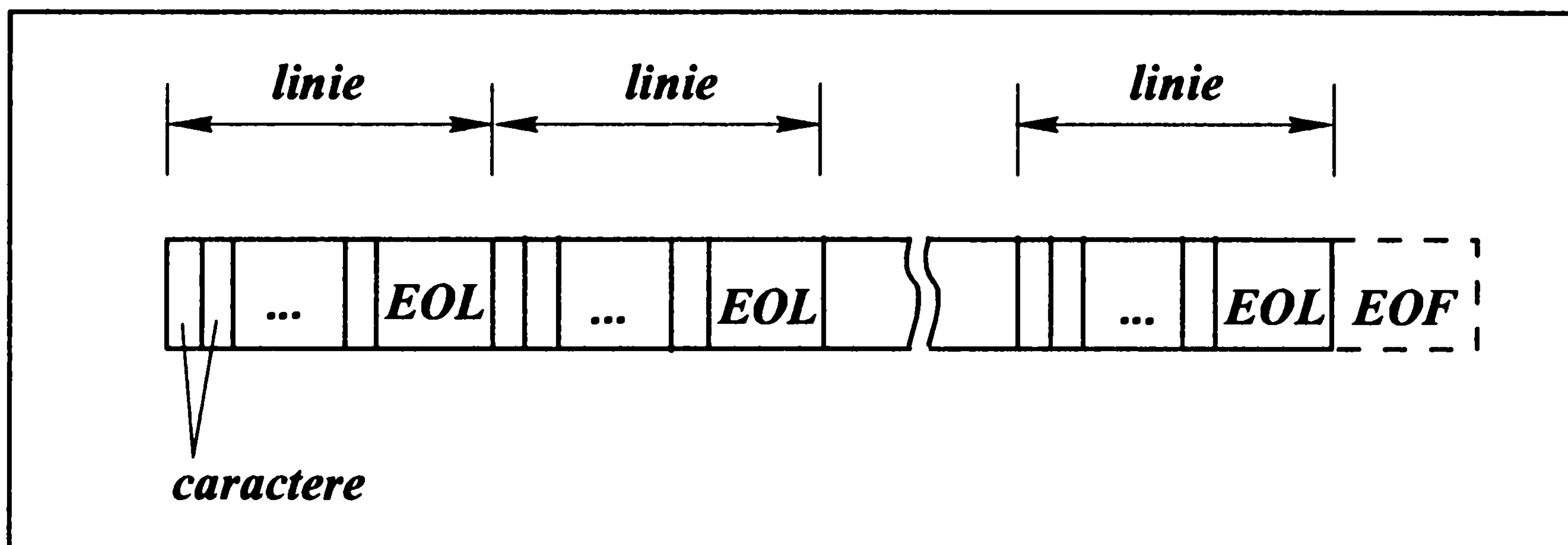


Fig. 4.9. Structura unui fișier *text*

tipul predefinit *text* fiind cunoscut oricărui program PASCAL. Subliniem faptul că tipurile *text* și **file of char** sînt distincte, întrucît fișierul **file of char** nu include elemente *EOL* (fig. 4.8).

Fișierele *text* pot fi prelucrate cu ajutorul procedurilor cunoscute, aplicabile oricărui tip de fișiere: *assign*, *reset*, *rewrite*, *read*,



write, close. În completare, limbajul include proceduri speciale, destinate prelucrării elementelor *EOL*:

writeln (*f*) — înscrie în fișier elementul *EOL* (sfârșit de linie);

readln (*f*) — trece la linia următoare.

Sfârșitul de linie este semnalat de funcția booleană *eofln* (*f*) care ia valoarea *true* după citirea ultimului caracter din linie.

O instrucțiune de forma

writeln (*f*, *e*<sub>1</sub>, *e*<sub>2</sub>, ..., *e*<sub>*n*</sub>)

este echivalentă cu secvența de instrucțiuni

write (*f*, *e*<sub>1</sub>, *e*<sub>2</sub>, ..., *e*<sub>*n*</sub>); writeln(*f*) .

O instrucțiune de forma

readln (*f*, *v*<sub>1</sub>, *v*<sub>2</sub>, ..., *v*<sub>*n*</sub>)

este echivalentă cu secvența de instrucțiuni

read (*f*, *v*<sub>1</sub>, *v*<sub>2</sub>, ..., *v*<sub>*n*</sub>); readln(*f*) .

Pentru introducerea și extragerea datelor, de regulă, se utilizează fișierele *text* predefinite Input și Output. Fișierul Input este destinat numai pentru operații de citire și este asociat cu fișierul de intrare al sistemului de operare (de regulă tastatura). Fișierul Output este destinat numai pentru operații de scriere și este asociat cu fișierul-standard de ieșire al sistemului de operare (de regulă ecranul). Aceste fișiere sînt deschise și închise automat la începutul și, respectiv, la sfârșitul execuției programului. Dacă numele fișierului nu este specificat în lista parametrilor unui apel de subalgoritm, se presupune că fișierul *text* implicit este Input sau Output, în funcție de natura subalgoritmului. De exemplu, read(*c*) este echivalent cu read(Input, *c*), iar write(*c*) este echivalent cu write(Output, *c*).

Pentru exemplificare prezentăm programul P93 care creează pe discul curent fișierul text FILE.TXT. Liniile fișierului sînt introduse de la tastatură (fișierul Input). Sfârșitul liniei se indică prin acționarea tastei <ENTER>, iar sfârșitul fișierului prin acționarea tastelor <CTRL+Z>, <ENTER>.

```
Program P93;
{ Crearea fișierului text FILE.TXT }
var F : text;
 c : char;
begin
 assign(F, 'FILE.TXT'); { asociază F cu FILE.TXT }
 rewrite(F); { deschide F pentru scriere }
```



```

while not eof do { eof(Input) }
 begin
 while not eoln do { eoln(Input) }
 begin
 read(c); { citește c din Input }
 write(F, c); { scrie c în F }
 end;
 writeln(F); { scrie EOL în F }
 readln; { trece la linia
 următoare din Input }

 end;
close(F); { închide F }
end.

```

Programul ce urmează afișează conținutul fișierului FILE.TXT pe ecran.

```

Program P94;
{ Citirea fișierului text FILE.TXT }
var F : text;
 c : char;
begin
 assign(F, 'FILE.TXT'); { asociază F cu FILE.TXT }
 reset(F); { deschide F pentru citire }
 while not eof(F) do
 begin
 while not eoln(F) do
 begin
 read(F, c); { citește c din F }
 write(c); { scrie c în Output }
 end;
 readln(F); { trece la linia următoare din F }
 writeln; { scrie EOL în Output }
 end;
 close(F); { închide F }
 readln;
end.

```

Explorarea caracter cu caracter a fișierelor *text* este greoaie atunci când secvențele de caractere din text trebuie interpretate ca formînd date de tip *integer*, *real*, *boolean*, *șir de caractere*. Conversia între forma externă și reprezentarea internă a acestor tipuri ar cădea în sarcina programatorului. În consecință, procedurile de citire/înscriere sînt extinse în felul următor.

În cazul fișierelor *text* variabila *v* dintr-un apel *read(f, v)* poate fi de tipul *integer*, *real*, *char* sau *șir de caractere*. La citire secvența de caractere care reprezintă variabila *v* va fi transformată în reprezentarea internă.

Expresia  $e$  dintr-un apel `write(f, e)` poate fi urmată de specificații de format. Valoarea expresiei poate fi de tipul `integer`, `real`, `boolean`, `char` sau *șir de caractere*. La scriere valoarea respectivă este transformată din reprezentarea internă într-o secvență de caractere.

Secvențele de caractere citite/scrise de procedurile `read/write` se conformează sintaxei constantelor de tipul variabilei/expresiei  $v/e$ .

Utilizarea procedurilor `write`, `writeln`, `read` și `readln` pentru prelucrarea fișierelor `Output` și `Input` a fost studiată în detalii în paragrafele 3.7 și 3.8. La fel se prelucrează orice fișier *text* definit de utilizator.

Pentru exemplificare prezentăm programul P95 care citește de la tastatură câte trei numere reale  $a$ ,  $b$ ,  $c$  pe care le scrie în fișierul `IN.TXT`. Apoi citind aceste trei numere, reprezentând laturile unui triunghi, scrie în fișierul `OUT.TXT` numerele  $a$ ,  $b$  și  $c$ , semiperimetrul  $p$  și aria triunghiului  $s$ . În continuare, conținutul fișierului `OUT.TXT` este afișat pe ecran.

```
Program P95;
{ Prelucrarea fișierelor IN.TXT și OUT.TXT }
var F, G : text;
 a, b, c, p, s : real;
 str : string;
begin
 assign(F, 'IN.TXT'); { asociază F cu IN.TXT }
 rewrite(F); { deschide F pentru scriere }
 writeln('Dați numerele reale a, b, c:');
 while not eof do
 begin
 readln(a, b, c); { citește a,b,c de la tastatură }
 writeln(F, a:8:2, b:8:2, c:8:2);
 { scrie a,b,c, în F }
 end;
 close(F); { închide F }
 reset(F); { deschide F
 pentru citire }
 assign(G, 'OUT.TXT'); { asociază G cu OUT.TXT }
 rewrite(G); { deschide G pentru scriere }
 while not eof(F) do
 begin
 readln(F, a, b, c); { citește a,b,c din F }
 write(G, a:8:2, b:8:2, c:8:2);
 { scrie a,b,c în G }

 p:=(a+b+c)/2;
 s:=sqrt(p*(p-a)*(p-b)*(p-c));
 writeln(G, p:15:2, s:15:4);
 { scrie p, s în G }
 end;
```



```

close(F); { închide F }
close(G); { închide G }
reset(G); { deschide G pentru citire }
while not eof(G) do
 begin
 readln(G, str); { citește str din G }
 writeln(str); { afișează str pe ecran }
 end;
close(G); { închide G }
readln;
end.

```

Pentru datele de intrare

```

1 1 1 <ENTER>
3 4 6 <ENTER>
<CTRL+Z><ENTER>

```

programul P95 afișează pe ecran:

```

1.00 1.00 1.00 1.50 0.4330
3.00 4.00 6.00 6.50 5.3327

```

## Întrebări și exerciții

- ❶ Care este diferența dintre un fișier *text* și un fișier **file of** char?
- ❷ Explicați semnificația elementelor *EOL* și *EOF*.
- ❸ Care este diferența dintre procedurile `read` și `readln`? Procedurile `write` și `writeln`?
- ❹ Lansați în execuție următorul program:

```

Program P96;
 { Asocierea fișierului FN cu consola }
type FisierNumere=file of integer;
var FN : FisierNumere;
 i : integer;
begin
 assign(FN, 'CON');
 rewrite(FN);
 i:=1;
 write(FN, i);
 i:=2;
 write(FN, i);
 i:=3;
 write(FN, i);
 close(FN);
 readln;
end.

```

Explicați rezultatele afișate pe ecran.

- ❶ Elaborați un program care afișează pe ecran conținutul oricărui fișier *text*.



- ⑥ Elaborați un program care afișează pe ecran numărul de vocale dintr-un fișier *text*.
- ⑦ Datele de intrare ale unui program sînt înmagazinate într-un fișier *text*. Fiecare linie a fișierului conține două numere întregi și trei numere reale separate prin spații. Elaborați un program care afișează suma numerelor întregi și suma numerelor reale din fiecare linie pe ecran.
- ⑧ Datele de intrare ale unui program sînt înmagazinate într-un fișier *text*. Fiecare linie a fișierului conține trei numere reale separate prin spații și unul din cuvintele ADMIS, RESPINS. Elaborați un program care:
- a) afișează conținutul fișierului în studiu pe ecran;
  - b) creează o copie de rezervă a fișierului;
  - c) creează un fișier *text* liniile căruia conțin media celor trei numere reale din liniile respective ale fișierului de intrare;
  - d) tipărește la imprimantă liniile fișierului de intrare, precedate de numerele de ordine 1, 2, 3 ș. a. m. d.
- ⑨ Fiecare linie a fișierului *text* conține următoarele date, separate prin spații:
- numărul de ordine (*integer*);  
numele (un *string* ce nu conține spații);  
nota la disciplina 1 (*real*);  
nota la disciplina 2 (*real*);  
nota la disciplina 3 (*real*).
- Elaborați un program care:
- a) creează o copie de rezervă a fișierului în studiu;
  - b) tipărește conținutul fișierului la imprimantă;
  - c) creează un fișier liniile căruia conțin următoarele date separate prin spații:
- numărul de ordine (*integer*);  
numele (*string*);  
nota medie (*real*).

Fișierul creat în punctul c trebuie afișat pe ecran și tipărit la imprimantă.

## Capitolul 5

.....

# FUNCȚII ȘI PROCEDURI

### 5.1. SUBPROGRAME

E cunoscut faptul că o problemă complexă poate fi rezolvată prin divizarea ei într-un set de părți mai mici (subprobleme). Pentru fiecare parte se scrie o anumită secvență de instrucțiuni, denumită **subprogram**. Problema în ansamblu se rezolvă cu ajutorul programului principal, în care pentru rezolvarea subproblemelor se folosesc apelurile subprogramelor respective. Când în programul principal se întâlnește un apel, execuția continuă cu prima instrucțiune din programul apelat (*fig. 5.1*). Când se termină executarea instrucțiunilor din subprogram, se revine la instrucțiunea imediat următoare apelului din programul principal.

În limbajul PASCAL există două tipuri de subprograme, și anume, funcții și proceduri:

$\langle \text{Subprograme} \rangle ::= \{ \langle \text{Funcție} \rangle; \mid \langle \text{Procedură} \rangle; \}$

**Funcțiile** sînt subprograme care calculează și returnează o valoare. Limbajul PASCAL conține un set de funcții predefinite, cunoscute oricărui program: `sin`, `cos`, `eof` etc. (vezi *tabelul 3.1*). În completare, programatorul poate defini funcții proprii, care se apelează în același mod ca funcțiile-standard. Prin urmare, conceptul de funcție extinde noțiunea de **expresie PASCAL**.

**Procedurile** sînt subprograme care efectuează prelucrarea datelor comunicate în momentul apelului. Limbajul conține procedurile predefinite `read`, `readln`, `write`, `writeln` ș.a., studiate în capitolele precedente. În completare, programatorul poate defini proceduri proprii, care se apelează în același mod ca procedurile-standard. Prin urmare, conceptul de procedură extinde noțiunea de **instrucțiune PASCAL**.

Subprogramele se definesc, în întregime, în partea declarativă a unui program (*fig. 3.14*). Evident, apelurile de funcții și proceduri se includ în partea executabilă a programului.

Un subprogram poate fi apelat chiar de el însuși, caz în care apelul este **recursiv**.

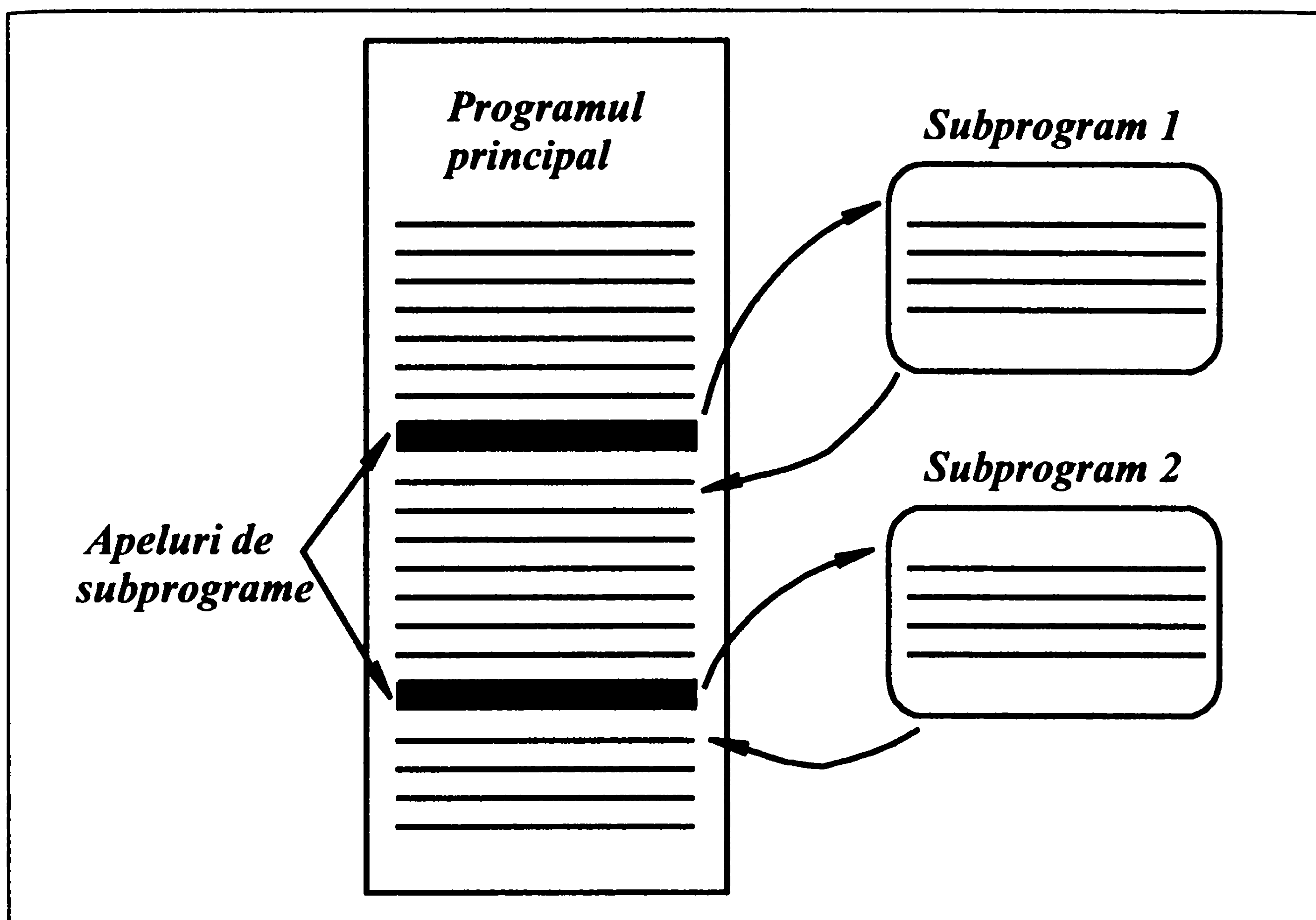


Fig. 5.1. Interacțiunea între program și subprogram

## Întrebări și exerciții

- ❶ Explicați termenii *program principal* și *subprogram*.
- ❷ Cum interacționează programul și subprogramul?
- ❸ Care este diferența dintre proceduri și funcții?
- ❹ Cum se apelează o funcție? În care instrucțiuni ale limbajului pot apărea apeluri de funcții?
- ❺ Cum se apelează o procedură?
- ❻ Numiți tipul argumentului și tipul rezultatului furnizat de funcțiile predefinite *abs*, *chr*, *eof*, *eoln*, *exp*, *ord*, *sin*, *sqr*, *sqrt*, *pred*, *succ*, *trunc*.
- ❼ Numiți tipul parametrilor actuali ai procedurilor *read* și *write*.
- ❽ Ce prelucrări de date efectuează procedurile *read* și *write*?

## 5.2. FUNCȚII

Textul PASCAL al unei **declarații de funcție** are forma:

```

function f (x_1 ; x_2 ...; x_n): t_r ;
 D ;
begin
 $f := e$;
 ...
end;

```



Prima linie este antetul funcției, format din:

$f$  — numele funcției;

$(x_1; x_2 \dots; x_n)$  — lista opțională de parametri formali reprezentând argumentele funcției;

$t_r$  — tipul rezultatului; acesta trebuie să fie numele unui tip simplu sau tip referință.

Antetul este urmat de **corpul funcției** format din declarațiile locale opționale  $D$  și instrucțiunea compusă **begin ... end**.

Declarațiile locale sînt grupate în secțiunile (eventual vide) **label**, **const**, **type**, **var**, **function/procedure**.

Numele  $f$  al funcției apare cel puțin o dată în partea stîngă a unei instrucțiuni de atribuire care se execută:  $f := e$ . Ultima valoare atribuită lui  $f$  va fi întoarsă în programul principal.

În mod obișnuit, un parametru formal din lista  $(x_1; x_2 \dots; x_n)$  are forma:

$(v_1, v_2 \dots, v_k) : t_p$

unde  $(v_1, v_2 \dots, v_k)$  sînt identificatori, iar  $t_p$  este un nume de tip.

Utilizarea funcției  $f$  se specifică printr-un apel de forma:

$f(a_1, a_2 \dots, a_n)$

unde  $(a_1, a_2 \dots, a_n)$  este lista de parametri actuali. De obicei, parametrii actuali sînt expresii, valorile cărora sînt comunicate funcției. Corespondența între un parametru actual și parametrul formal se face prin poziția ocupată de aceștia în cele două liste. Parametrul actual trebuie să fie **compatibil** din punct de vedere al atribuirii cu tipul parametrului formal.

*Exemplu:*

```
Program P97;
{ Declararea și utilizarea funcției Putere }
type Natural=0..MaxInt;
var a : real;
 b : Natural;
 c : real;
 s : integer;
 t : integer;
 v : real;

function Putere(x : real; n : Natural) : real;
 { calcularea lui x la puterea n }
 var p : real;
 i : integer;
 begin
 p:=1;
 for i:=1 to n do p:=p*x;
 Putere:=p;
```

```

end; { Putere }

begin
 a:=3.0;
 b:=2;
 c:=Putere(a, b);
 writeln(a:10:5, b:4, c:10:5);

 s:=2;
 t:=4;
 v:=Putere(s, t);
 writeln(s:5, t:4, v:10:5);

 readln;
end.

```

Funcția `Putere` are doi parametri formali: `x` de tipul `real` și `n` de tipul `Natural`. Funcția returnează o valoare de tipul `real`. În corpul funcției sînt declarate variabilele locale `p` și `i`.

La execuția apelului `Putere(a, b)` valorile 3.0 și 2 ale parametrilor actuali `a`, `b` se transmit parametrilor formali, respectiv, `x` și `n`. De menționat că tipul lui `a` coincide cu tipul lui `x` și tipul lui `b` coincide cu tipul lui `n`.

În cazul apelului `Putere(s, t)` tipul parametrilor actuali `s`, `t` nu coincide cu tipul parametrilor formali, respectiv, `x` și `n`. Totuși, apelul este corect, întrucît tipurile respective sînt compatibile din punct de vedere al atribuirii.

## Întrebări și exerciții

- ❶ Se consideră următoarea declarație:

```

function Factorial(n : integer) : integer;
var p, i : integer;
begin
 p:=1;
 for i:=1 to n do p:=p * i;
 Factorial:=p;
end;

```

Numiți tipul parametrului formal și tipul rezultatului returnat de funcție. Precizați variabilele declarate în corpul funcției.

Elaborați un program care afișează pe ecran valorile  $n!$  pentru  $n = 2, 3$  și  $7$ .

- ❷ În care loc al programului principal se includ declarațiile de funcții?
- ❸ Comentați programul ce urmează:

```

Program P98;
{ Eroare }
function Factorial(n : 0..7) : integer;
var p, i : integer;

```

```

begin
 p:=1;
 for i:=1 to n do p:=p*i;
 Factorial:=p;
end; { Factorial }

begin
 writeln(Factorial(4));
 readln;
end.

```

④ Se consideră antetul

```

function F(x : real; y : integer;
 z : char) : boolean;

```

Care din apelurile ce urmează sînt corecte:

- a) F(3.18, 4, 'a');
- b) F(4, 4, '4');
- c) F(4, 4, 4);
- d) F(4, 3.18, 'a');
- e) F(3.18, 4, 4);
- f) F('3.18', 4, '4');
- g) F(15, 21, '3');
- h) F(15, 21, 3).

⑤ Elaborați o funcție care calculează:

- a) suma numerelor reale  $a, b, c, d$ ;
- b) media numerelor întregi  $i, j, k, m$ ;
- c) minimumul din numerele  $a, b, c, d$ ;
- d) numărul de vocale într-un șir de caractere;
- e) numărul de consoane într-un șir de caractere;
- f) rădăcina ecuației  $ax + b = 0$ ;
- g) cel mai mic divizor al numărului întreg  $n > 0$ , diferit de 1;
- h) cel mai mare divizor comun al numerelor naturale  $a, b$ ;
- i) cel mai mic multiplu comun al numerelor naturale  $a, b$ ;
- j) ultima cifră în notația zecimală a numărului întreg  $n > 0$ ;
- k) cîte cifre sînt în notația zecimală a numărului întreg  $n > 0$ ;
- l) cifra superioară în notația zecimală a numărului întreg  $n > 0$ ;
- m) numărul de apariții a caracterului dat într-un șir de caractere.

⑥ Se consideră următoarele declarații:

```

const nmax=100;
type Vector=array [1..nmax] of real;

```

Elaborați o funcție care calculează :

- a) suma componentelor unui vector;
- b) media componentelor vectorului;
- c) componenta maximă;
- d) componenta minimă.



⑦ Se consideră următoarele tipuri de date:

```
type Punct= record
 x, y : real
 end;
 Segment= record
 A, B : Punct
 end;
 Triunghi= record
 A, B, C : Punct
 end;
 Dreptunghi=record
 A, B, C, D : Punct
 end;
 Cerc= record
 Centru : Punct;
 Raza : real
 end;
```

Elaborați o funcție care calculează:

- a) lungimea segmentului;
- b) lungimea cercului;
- c) aria cercului;
- d) aria triunghiului;
- e) aria dreptunghiului.

⑧ Variabila *A* este introdusă prin declarația

```
var A : set of char;
```

Elaborați o funcție care returnează numărul de caractere din mulțimea *A*.

- ⑨ Elaborați o funcție care să calculeze diferența în secunde între două momente de timp date prin oră, minute și secunde.
- ⑩ Un triunghi este definit prin coordonatele vîrfurilor sale. Scrieți funcții care, pentru două triunghiuri date, să studieze dacă:
  - a) au aceeași arie;
  - b) sînt asemenea;
  - c) primul este în interiorul celui de-al doilea.

### 5.3. PROCEDURI

Forma generală a textului unei declarații de procedură este:

```
procedure p ($x_1; x_2 \dots; x_n$);
D;
begin
 ...
end;
```

În antetul procedurii apar:

$p$  — numele procedurii;

$(x_1; x_2 \dots; x_n)$  — lista opțională de parametri formali;

În **corpul procedurii** sînt incluse:

$D$  — declarațiile locale (opționale) grupate după aceleași reguli ca în cazul funcțiilor;

**begin ... end** — instrucțiune compusă; ea nu conține vreo atribuire asupra numelui procedurii.

Procedura poate să întoarcă mai multe rezultate, dar nu prin numele ei, ci prin variabile desemnate special (cu prefixul **var**) în lista de parametri formali.

Parametrii din listă introduși prin declarații de forma

$v_1, v_2 \dots, v_k: t_p$

se numesc **parametri-valoare**. Aceștia servesc pentru transmiterea de valori din programul principal în procedură.

Parametrii formali introduși în listă prin declarații de forma

**var**  $v_1, v_2 \dots, v_k: t_p$

se numesc **parametri-variabilă** și servesc pentru întoarcerea rezultatelor din procedură în programul principal.

Activarea unei proceduri se face printr-un apel de forma

$p(a_1, \dots, a_n)$

unde  $(a_1, \dots, a_n)$  este lista de **parametri actuali**. Spre deosebire de funcție, apelul de procedură este o instrucțiune; aceasta se inserează în programul principal în locul în care sînt dorite efectele produse de execuția procedurii.

În cazul unui **parametru-valoare** drept parametru actual poate fi utilizată orice expresie de tipul respectiv, în particular o constantă sau o variabilă. Modificările parametrilor valoare nu se transmit în exteriorul subprogramului.

În cazul unui **parametru-variabilă** drept parametri actuali pot fi utilizate numai variabile. Evident, modificările parametrilor în studiu vor fi transmise programului apelant.

*Exemplu:*

```
Program P99;
 { Declararea și utilizarea procedurii Lac }
var a, b, c,
 t, q : real;

procedure Lac(r : real; var l, s : real);
 { lungimea și aria cercului }
 { r - raza; l - lungimea; s - aria }
```

```

const Pi=3.14159;
begin
 l:=2*Pi*r;
 s:=Pi*sqr(r);
end; { Lac }

begin
 a:=1.0;
 Lac(a, b, c);
 writeln(a:10:5, b:10:5, c:10:5);

 Lac(3.0, t, q);
 writeln(3.0:10:5, t:10:5, q:10:5);

 readln;
end.

```

Procedura Lac are trei parametri formali:  $r$ ,  $l$  și  $s$ . Parametrul  $r$  este un parametru-valoare, iar  $l$  și  $s$  sînt parametri-variabilă.

Execuția instrucțiunii  $Lac(a, b, c)$  determină transmiterea valorii 1.0 drept valoare a parametrului formal  $r$  și a locațiilor (adrese-  
lor) variabilelor  $b$  și  $c$  drept locații (adrese) ale parametrilor formali  $l$  și  $s$ . Prin urmare, secvența de instrucțiuni

```

a:=1.0;
Lac(a, b, c)

```

este echivalentă cu secvența

```

b:=2*Pi*1.0;
c:=Pi*sqr(1.0).

```

În mod similar, instrucțiunea

```

Lac(3.0, t, q)

```

este echivalentă cu secvența

```

t:=2*Pi*3.0;
q:=Pi*sqr(3.0).

```

## Întrebări și exerciții

- ❶ Care este diferența dintre un *parametru-valoare* și un *parametru-variabilă*?
- ❷ Se consideră declarațiile:

```

Var k, m, n : integer;
 a, b, c : real;
procedure P(i : integer; var j : integer;
 x : real; var y : real);
begin
 { ... }
end.

```



Care din apelurile ce urmează sînt corecte?

- |                     |                    |
|---------------------|--------------------|
| a) P(k, m, a, b);   | f) P(n, m, 6, b);  |
| b) P(3, m, a, b);   | g) P(n, m, 6, 20); |
| c) P(k, 3, a, b);   | h) P(a, m, b, c);  |
| d) P(m, m, a, b);   | i) P(i, i, i, i);  |
| e) P(m, k, 6.1, b); | j) P(a, a, a, a).  |

Argumentați răspunsul.

③ Comentați programul ce urmează:

```
Program P100;
 { Eroare }
var a : real;
 b : integer;
procedure P(x : real; var y : integer);
begin
 { ... }
end; { P }
begin
 P(a, b);
 P(0.1, a);
 P(1, b);
 P(a, 1);
end.
```

④ Ce va afișa pe ecran programul ce urmează?

```
Program P101;
 { Parametru-valoare și parametru-variabilă }
var a, b : integer;

procedure P(x : integer; var y : integer);
begin
 x:=x+1;
 y:=y+1;
 writeln('x=', x, ' y=', y);
end; { P }

begin
 a:=0;
 b:=0;
 P(a, b);
 writeln('a=', a, ' b=', b);
 readln;
end.
```

Argumentați răspunsul.

⑤ Elaborați o procedură care:

- a) calculează rădăcinile ecuației  $ax^2 + bx + c = 0$ ;
  - b) radiază dintr-un șir caracterul indicat în apel;
  - c) încadrează un șir de caractere între simbolurile “#”;
  - d) ordonează componentele unui tablou **array**[ 1..100] **of** real în ordine crescătoare;
  - e) ordonează componentele unui fișier **file of** integer în ordine descrescătoare;
  - f) calculează și depune într-un tablou numerele prime mai mici decât un număr natural dat  $n$ .
- ⑥ În exercițiul 4 din paragraful 4.3 este definit tipul de date **ListaPersoane**. Elaborați procedurile necesare pentru calcularea datelor specificate în punctele a, b, c, ..., k ale exercițiului în studiu.
- ⑦ Elaborați o procedură care:
- a) creează o copie de rezervă a unui fișier *text*;
  - b) exclude dintr-un fișier *text* liniile vide;
  - c) numerotează liniile unui fișier *text*;
  - d) concatenează două fișiere *text* într-unul singur;
  - e) concatenează  $n$  fișiere *text* ( $n > 2$ ) într-unul singur.
- ⑧ În exercițiul 5 din paragraful 4.9 este definit tipul de date **FisierAngajati**. Elaborați procedurile necesare pentru prelucrarea datelor conform specificărilor din punctele a, b, c, ..., h ale exercițiului în studiu.
- ⑨ Să se definească un tip de date pentru numere naturale foarte mari și să se scrie proceduri care să adune și să scadă astfel de numere.

## 5.4. PARAMETRI FORMALI *FUNCȚIE* / *PROCEDURĂ*

În completare la parametrii-valoare și parametrii-variabilă, limbajul PASCAL permite utilizarea *parametrilor-funcție* și *parametrilor-procedură*.

Specificarea unui parametru formal *funcție/procedură* are forma antetului acestei funcții/proceduri. De exemplu, în declarația

```
procedure Q(function F(x:real):real,
 i : integer);
begin
 { ... }
end;
```

apare ca parametru-*funcție* antetul **function** F(x:real):real.

Un apel al procedurii Q ar putea fi

Q(F1, j),

unde F1 este numele unei funcții cunoscute în locul apelului.

În Turbo PASCAL parametrii-*funcție/procedură* se declară explicit ca aparținând unui tip procedural.

**Tipul procedural** se definește cu ajutorul cuvîntului-cheie **type**. Definiția respectivă specifică numele și tipul parametrilor formali ai unui subprogram. În cazul funcțiilor se indică și tipul valorii returnate.

*Exemplu:*

```
type TF=function(x : real) : real;
 TP=procedure(i, j : integer);
var v : TF;
 p : TP;
```

Mulțimea valorilor tipului TF este formată din denumirile tuturor funcțiilor PASCAL de o variabilă reală cu rezultate reale. Mulțimea valorilor tipului TP este formată din denumirile tuturor procedurilor PASCAL de două variabile întregi.

Menționăm că la definirea tipurilor procedurale nu apare nici un fel de nume de funcție/procedură.

Asupra variabilelor de tip procedural pot fi efectuate atribuiri. De exemplu, în prezența declarațiilor

```
function F1(x : real) : real;
begin
 { ... }
end;

procedure P1(m, n : integer);
begin
 { ... }
end;
```

atribuirile

```
v:=F1;
p:=P1
```

sînt corecte.

Folosirea tipurilor procedurale simplifică declararea parametrilor formali *funcție/procedură*. Un astfel de parametru se specifică acum ca un caz particular de parametru-valoare printr-un **nume** urmat de un **tip procedural**. Parametrul actual corespunzător va fi un nume de *funcție/procedură* de tip compatibil celui procedural.

*Exemplu:*

```
Program P102;
{ Parametru formal funcție }

type TF=function(x : real) : real;
var v : TF;

function F1(x : real) : real;
```



```

begin
 F1:=x
end; { F1 }

function F2(x : real) : real;
begin
 F2:=3*x+2
end; { F2 }

function F3(x : real) : real;
begin
 F3:=sqr(x)+2*x+1
end; { F3 }

procedure Tabel(F : TF; x1, x2, dx : real);
{ Tabelarea funcției F(x) }
var a, b : real;
begin
 a:=x1;
 while a<=x2 do
 begin
 b:=F(a);
 writeln(a:15:5, b:15:5);
 a:=a+dx;
 end;
end; { Tabel }

begin
 writeln('Tabelul funcției F1');
 Tabel(F1, 0, 2, 0.5);

 writeln('Tabelul funcției F2');
 Tabel(F2, 0, 2, 0.5);

 writeln('Tabelul funcției F3');
 v:=F3;
 Tabel(v, 0, 2, 0.5);
 readln;
end.

```

În programul în studiu sînt definite:

- tipul procedural TF;
- variabila v de tipul TF;
- funcțiile F1, F2 și F3 de tipul TF;
- procedura Tabel cu parametrul-funcție F și parametrii-valoare x1, x2 și dx.

Procedura Tabel afișează pe ecran tabelul oricărei funcții de tipul TF. În corpul programului această procedură este apelată pentru tabe-

larea funcțiilor F1, F2 și F3. În ultimul apel numele funcției F3 se transmite procedurii Tabel prin intermediul variabilei v.

Subliniem faptul că funcțiile și procedurile predefinite nu pot fi transmise ca parametri actuali. De exemplu, apelul procedurii Tabel (sin, ...) nu este corect, deoarece sin este o funcție predefinită. Inconvenientul poate fi însă depășit definind o funcție fsin

```
function fsin(x : real) : real;
begin
 fsin:=sin(x);
end; { fsin}
```

care poate fi folosită în apelul Tabel (fsin, ...) cu efectul dorit.

## Întrebări și exerciții

- ❶ Cum se specifică un parametru formal *funcție/procedură* în PASCAL? În Turbo PASCAL?
- ❷ Cum se definește un tip procedural? Care este mulțimea de valori ale unui tip procedural?
- ❸ Corectați programul:

```

Program P103;
{ Eroare }
type TF=function(x : real) : real;

function F1(x : real) : real;
begin
 F1:=sqr(x);
end; { F1 }

procedure T(F : TF; x : real);
begin
 writeln('x=', x, ' f(x)=', F(X));
end; { T }

begin
 T(F1, 2);
 T(sqrt, 2);
 readln;
end.

```

- 4** Utilizând procedura Tabel din programul P102, elaborați un program care tablează funcțiile:
- |                                         |                                                  |
|-----------------------------------------|--------------------------------------------------|
| a) $x^3 - 4$ , $2x + 1$ , $6x$ ;        | c) $\sin x$ , $\cos x$ , $\operatorname{tg} x$ ; |
| b) $x^2 + x - 1$ , $x + 1$ , $\sin x$ ; | d) $x$ , $x^2$ , $x^3$ .                         |
- 5** Precizați ce va afișa pe ecran programul ce urmează:

```
Program P104;
 { Parametru formal procedură }
```

```

type TP=procedure (i : integer);
var v : TP;

procedure P1(j : integer);
begin
 writeln(j+1);
end; { P1 }

procedure P2(k : integer);
begin
 writeln(k+2);
end; { P2 }

procedure P3(m : integer);
begin
 writeln(m+3);
end; { P3 }

procedure T(P : TP; n : integer);
begin
 P(n);
end; { T }

begin
 v:=P3;
 T(v, 1);
 v:=P2;
 T(v, 1);
 T(P1, 1);
 readln;
end.

```

⑥ Elaborați o procedură care afișează pe ecran tabelele funcțiilor de două variabile reale cu rezultate reale. Tabelați funcțiile:

- a)  $z = x + y, \quad z = x - y, \quad z = y - x;$
- b)  $z = \cos(x + y), \quad z = \sin(x + y), \quad z = \operatorname{tg}(x + y);$
- c)  $z = \begin{cases} x + y, & x \geq y; \\ x - y, & x < y; \end{cases}$
- $z = \begin{cases} x^2 + y^2, & x + y \geq 2; \\ x^2 - y^2, & x + y < 2; \end{cases}$
- $z = \begin{cases} \sin(x + y), & x \geq y; \\ \cos(x - y), & x < y; \end{cases}$

⑦ Funcția  $y = f(x)$  este nenulă în punctele  $1, 2, \dots, n$ . Elaborați o procedură care determină numărul schimbărilor de semn în șirul  $f(1), f(2), \dots, f(n)$ .

⑧ Se consideră mulțimea tuturor funcțiilor  $f(i)$  de o variabilă întreagă cu rezultate reale. Elaborați o funcție care calculează  $f(1) + f(2) + \dots + f(n)$ .

⑨ Se consideră funcția  $f(i)$  de o variabilă întreagă cu rezultate întregi. Elaborați o funcție care calculează de câte ori șirul  $f(1), f(2), \dots, f(n)$  conține subșirul 1, 0, 1.



- ⑩ Ecuația  $f(x) = 0$  are în intervalul  $(a, b)$  o singură rădăcină. Elaborați o procedură pentru determinarea rădăcinii ecuației în studiu prin metoda înjumătățirii.

*Indicație:* Se calculează  $c = (a + b)/2$ . Analizând semnele valorilor  $f(a)$ ,  $f(c)$  și  $f(b)$  se selectează unul din intervalele  $(a, c)$ ,  $(c, b)$  ce conține rădăcina ecuației. Intervalul selectat se împarte din nou în două intervale ș.a.m.d. până se ajunge la precizia dorită.

## 5.5. DOMENII DE VIZIBILITATE

Corpul unui program sau subprogram se numește **bloc**. Deoarece subprogramele sînt incluse în programul principal și pot conține la rîndul lor alte subprograme, rezultă că blocurile pot fi **imbricate** (in-

```
Program P105;
{ Structura de bloc a programului } { nivel 0 }
var a: real;
{ 1 }
 procedure P(b : real);
 var c : real; { nivel 1 }
 { 2 }
 procedure Q(d : integer);
 { 3 } { nivel 2 }
 var c : char;
 { 4 }
 begin
 c:=chr(d);
 writeln('În procedura Q c = ', c);
 end; { 5 }
 begin
 writeln('b=', b);
 c:=b+1;
 writeln('În procedura P c = ', c);
 Q(35);
 end; { 6 }

function F(x : real) : real;
begin
 f:=x/2;
end;

begin
 a:=F(5);
 writeln('a=', a);
 P(a);
 readln;
end { 7 } .
```

Fig. 5.2. Structura de bloc a unui program PASCAL

cluse unul în altul). Această imbricare de blocuri este denumită **structura de bloc** a programului PASCAL.

Într-o structură fiecărui bloc  $i$  se atașează câte un nivel de imbricare. Programul principal este considerat de nivel 0, un bloc definit în programul principal este de nivel 1. În general, un bloc definit în nivelul  $n$  este de nivelul  $n + 1$ .

Pentru exemplificare, în *fig. 5.2* este prezentată structura de bloc a programului P105.

De regulă, un bloc PASCAL include declarații de etichete, variabile, funcții, parametri ș.a.m.d. O declarație introduce un nume, care poate fi o etichetă sau un identificator. O declarație dintr-un bloc poate redefini un nume declarat în exteriorul lui. În consecință, în diferite părți ale programului unul și același nume poate desemna obiecte diferite.

Prin **domeniul de vizibilitate** al unei declarații se înțelege textul de program, în care numele introdus desemnează obiectul specificat de declarația în studiu. Domeniul de vizibilitate începe imediat după terminarea declarației și se sfârșește o dată cu textul blocului respectiv. Deoarece blocurile pot fi imbricate, domeniul de vizibilitate nu este neapărat o porțiune continuă din textul programului. Domeniul de vizibilitate al unei declarații dintr-un bloc inclus **acoperă** domeniul de vizibilitate al declarației ce implică același nume din blocul exterior.

De exemplu, în programul P105 domeniul de vizibilitate al declarației **var**  $a : \text{integer}$  este textul cuprins între punctele marcate { 1} și { 7}. Domeniul de vizibilitate al declarației **var**  $c : \text{real}$  este format din două fragmente de text cuprinse între { 2}, { 3} și { 5}, { 6}. Domeniul de vizibilitate al declarației **var**  $c : \text{char}$  este textul cuprins între { 4} și { 5}.

Cunoașterea domeniilor de vizibilitate ale declarațiilor este necesară pentru determinarea obiectului curent desemnat de un nume.

De exemplu, identificatorul  $c$  din instrucțiunea

$c := \text{chr}(d)$

a programului P105 desemnează o variabilă de tip  $\text{char}$ . Același identificator din instrucțiunea

$c := b + 1$

desemnează o variabilă de tip  $\text{real}$ .

De reținut că declarația unui nume de funcție/procedură se consideră terminată la sfârșitul antetului. Prin urmare, domeniul de vizibilitate al unei astfel de declarații include și corpul funcției/procedurii respective. Acest fapt face posibil **apelul recursiv**: în corpul funcției/procedurii aceasta poate fi referită, fiind vizibilă. Evident, decla-



rația unui parametru formal este vizibilă numai în corpul subprogramului respectiv.

De exemplu, domeniul de vizibilitate al declarației **procedure** Q este textul cuprins între punctele marcate { 3} și { 6} . Domeniul de vizibilitate al declarației `d: integer` este textul cuprins între { 3} și { 5} .

## Întrebări și exerciții

- ❶ Cum se determină domeniul de vizibilitate al unei declarații?
- ❷ Determinați domeniile de vizibilitate ale declarațiilor `b : real` și `x : real` din programul P105 (fig 5.2).
- ❸ Precizați structura de bloc a programului ce urmează. Indicați domeniul de vizibilitate al fiecărei declarații și determinați obiectele desemnate de fiecare apariție a identificatorilor `c` și `x`.

```
Program P106;
{ Redefinirea constantelor }
const c=1;

function F1(x : integer) : integer;
begin
 F1:=x+c;
end; { F1 }

function F2(c : real) : real;
const x=2.0;
begin
 F2:=x+c;
end; { F2 }

function F3(x : char) : char;
const c=3;
begin
 F3:=chr(ord(x)+c);
end; { F3 }
begin
 writeln('F1=', F1(1));
 writeln('F2=', F2(1));
 writeln('F3=', F3('1'));
 readln;
end.
```

Ce va afișa pe ecran programul în studiu?

- ❹ Determinați domeniile de vizibilitate ale identificatorilor `P` și `F` din programul P105 (fig. 5 2).
- ❺ Comentați programul ce urmează:

```
Program P107;
{ Eroare }
var a : real;
```



```

procedure P(x : real);
var a : integer;
begin
 a:=3.14;
 writeln(x+a);
end; { P }
begin
 a:=3.14;
 P(a);
end.

```

- ⑥ Cum se determină obiectul desemnat de apariția unui nume într-un program PASCAL?

## 5.6. COMUNICAREA PRIN VARIABILE GLOBALE

Execuția unui apel de subprogram presupune transmiterea datelor de prelucrat funcției sau procedurii respective. După executarea ultimei instrucțiuni din subprogram, rezultatele produse trebuie întoarse în locul de apel. Cunoaștem deja că datele de prelucrat și rezultatele produse pot fi transmise prin parametri. Parametrii formali se specifică în antetul funcției sau procedurii, iar parametrii actuali — în locul apelului.

În completare la modul de transmitere a datelor prin parametri, limbajul PASCAL permite comunicarea prin variabile globale.

O variabilă este globală **relativ la un subprogram** atunci când ea este declarată în programul sau subprogramul ce îl cuprinde fără să fie redeclarată în subprogramul în studiu. Întrucât variabilele globale sînt cunoscute atît în subprogram, cît și în afara lui, ele pot fi folosite pentru transmiterea datelor de prelucrat și returnarea rezultatelor.

*Exemplu:*

```

Program P108;
 { Comunicarea prin variabile globale }
var a, { variabilă globală în P }
 b : real; { variabilă globală în P, F }

procedure P;
var c : integer; { variabilă locală în P }
begin
 c:=2;
 b:=a*c;
end; { P }

function F : real;
var a : 1..5; { variabilă locală în F }

```

```

begin
 a:=3;
 F:=a+b;
end; { F }

begin
 a:=1;
 P;
 writeln(b); { se afișează 2.000000000000E+00 }
 writeln(F); { se afișează 5.000000000000E+00 }
 readln;
end.

```

Datele de prelucrat se transmit procedurii P prin variabila globală a. Rezultatul produs de procedură se returnează în blocul de apel prin variabila globală b. Valoarea argumentului funcției F se transmite prin variabila globală b. Menționăm că variabila a este locală în F și nu poate fi folosită pentru transmiterea datelor în această funcție.

De obicei, comunicarea prin variabile globale se utilizează în cazurile în care mai multe subprograme prelucrează aceleași date. Pentru exemplificare amintim funcțiile cu argumente de tip *tablou*, procedurile care prelucrează tablouri și fișiere de angajați, persoane, elevi etc.

## Întrebări și exerciții

- ❶ Explicați termenii *variabilă globală relativ la un subprogram* și *variabilă locală într-un subprogram*.
- ❷ Numiți variabilele globale și variabilele locale din programul P105 (fig. 5.2).
- ❸ Poate fi oare o variabilă locală în același timp și o variabilă globală relativ la un subprogram?
- ❹ Numiți variabilele globale și variabilele locale din programul ce urmează. Ce va afișa pe ecran acest program?

```

Program P109;
 { Comunicarea prin variabile globale }
var a : integer;

procedure P;
var b, c, d : integer;

procedure Q;
begin
 c:=b+1;
end; { Q }

procedure R;
begin
 d:=c+1;
end; { R }

```



```

begin
 b:=a;
 Q;
 R;
 a:=d;
end; { P }

```

```

begin
 a:=1;
 P;
 writeln(a);
 readln;
end.

```

⑤ Se consideră declarațiile

```

Type Ora=0..23;
 Grade=-40..+40;
 Temperatura=array[Ora] of Grade;

```

Componentele unei variabile de tip *Temperatura* reprezintă temperaturile măsurate din oră în oră pe parcursul a 24 de ore. Elaborați o procedură care:

- indică maximumul și minimumul temperaturii;
- indică ora (orele) la care s-a înregistrat o temperatură maximă;
- înscrie ora (orele) la care s-a înregistrat o temperatură minimă într-un fișier *text*.

Comunicarea cu procedurile respective se va face prin variabile globale.

⑥ Se consideră fișiere arbitrare de tip *text*. Elaborați o funcție care:

- returnează numărul de linii dintr-un fișier;
- calculează numărul de vocale dintr-un text;
- calculează numărul de cuvinte dintr-un text (cuvintele reprezintă șiruri de caractere separate prin spațiu sau sfârșit de linie);
- returnează lungimea medie a liniilor din text;
- calculează lungimea medie a cuvintelor din text;
- returnează numărul semnelor de punctuație din text.

Comunicarea cu funcțiile respective se va face prin variabile globale.

## 5.7. EFECTE COLATERALE

Destinația unei funcții este să întoarcă ca rezultat o singură valoare. În mod obișnuit, argumentele se transmit funcției prin parametri-valoare, iar rezultatul calculat se returnează în locul de apel prin numele funcției. În completare, limbajul PASCAL permite transmiterea argumentelor prin variabile globale și parametri-variabilă.

Prin **efect colateral** se înțelege o atribuire (în corpul funcției) a unei valori la o variabilă globală sau la un parametru formal variabilă.



Efectele colaterale pot influența în mod neașteptat execuția unui program și complică procesele de depanare.

Prezentăm în continuare exemple defectuoase de programare care folosesc funcții cu efecte colaterale.

```
Program P110;
{ Efect colateral - atribuire la o variabilă globală }
var a : integer; { variabilă globală }

function F(x : integer) : integer;
 begin
 F:=a*x;
 a:=a+1; { atribuire defectuoasă }
end; { F }

begin
 a:=1;
 writeln(F(1)); { se afișează 1 }
 writeln(F(1)); { se afișează 2 }
 writeln(F(1)); { se afișează 3 }
 readln;
end.
```

În programul P110 funcția F returnează valoarea expresiei  $a * x$ . Pe lângă asta însă, atribuirea  $a := a + 1$  alterează valoarea variabilei globale a. În consecință, pentru una și aceeași valoare 1 a argumentului x funcția returnează rezultate diferite, fapt ce nu se încadrează în conceptul uzual de funcție.

```
Program P111;
{ Efect colateral - atribuire la un parametru formal }
var a : integer;

function F(var x : integer) : integer;
 begin
 F:=2*x;
 x:=x+1; { atribuire defectuoasă }
end; { F }

begin
 a:=2;
 writeln(F(a)); { se afișează 4 }
 writeln(F(a)); { se afișează 6 }
 writeln(F(a)); { se afișează 8 }
 readln;
end.
```

În programul P111 funcția F returnează valoarea expresiei  $2 * x$ . Întrucât x este un parametru formal variabilă, atribuirea  $x := x + 1$  schimbă valoarea parametrului actual din apel, și anume, a variabilei

a din programul principal. Faptul că apelurile textual identice  $F(a)$ ,  $F(a)$  și  $F(a)$  returnează rezultate ce diferă, poate crea confuzii în procesul depanării.

În cazul procedurilor, atribuirile asupra variabilelor globale produc efecte colaterale similare celor discutate pentru astfel de atribuiri la funcții. Întrucât mijlocul-standard de întoarcere de rezultate din procedură este prin parametri formali variabilă, atribuirile asupra unor astfel de parametri nu sînt considerate ca efecte colaterale.

Efectele colaterale introduc abateri de la procesul-standard de comunicare, prin care variabilele participante sînt desemnate explicit ca parametri formali în declarație și parametri actuali în apel. Consecințele efectelor colaterale se pot propaga în domeniul de vizibilitate al declarațiilor globale și pot interfera cu cele similare produse la execuția altor proceduri și funcții. În astfel de condiții, utilizarea variabilelor globale devine riscantă. Prin urmare, la elaborarea programelor complexe se vor aplica următoarele recomandări:

- ✓ Comunicarea funcțiilor cu mediul de chemare se va face prin transmiterea de date spre funcție prin parametri formali valoare și întoarcerea unui singur rezultat prin numele ei.
- ✓ Comunicarea procedurilor cu mediul de chemare se va face prin transmiterea de date prin parametri formali valoare sau variabilă și întoarcerea rezultatelor prin parametri formali variabilă.
- ✓ Variabilele globale pot fi folosite pentru transmiterea datelor în subprograme, însă valorile lor nu trebuie să fie schimbate de acestea.

## Întrebări și exerciții

- ❶ Care este cauza efectelor colaterale? Ce consecințe pot avea aceste efecte?
- ❷ Precizați ce vor afișa pe ecran programele ce urmează:

```
Program P112;
{ Efecte colaterale }
var a, b : integer;

 function F(x : integer) : integer;
 begin
 F:=a*x;
 b:=b+1;
 end; { F }

 function G(x : integer) : integer;
 begin
 G:=b+x;
 a:=a+1;
 end; { G }
```

```

begin
 a:=1; b:=1;
 writeln(F(1));
 writeln(G(1));
 writeln(F(1));
 writeln(G(1));
 readln;
end.

```

```

Program P113;
{ Efecte colaterale }
var a : integer;
 b : real;

 function F(var x : integer) : integer;
 begin
 F:=x;
 x:=x+1;
 end; { F }

 procedure P(x,y:integer; var z:real);
 begin
 z:=x/y;
 end; { P }

```

```

begin
 a:=1;
 P(F(a), a, b);
 writeln(a, ' ', b);
 readln;
end.

```

```

Program P114;
{ Efecte colaterale }
var a, b : real;

 procedure P(var x, y : real);
 { Interschimbarea valorilor variabilelor x, y }
 begin
 a:=x;
 x:=y;
 y:=a;
 end; { P }

begin
 a:=1; b:=2;
 P(a, b);
 writeln(a, b);
 a:=3; b:=4;
 P(a, b);

```



```
writeln(a, b);
readln;
end.
```

❸ Cum pot fi evitate efectele colaterale?

## 5.8. RECURSIA

**Recursia** se definește ca o situație în care un subprogram se autoapelează fie direct, fie prin intermediul altei funcții sau proceduri. Subprogramul care se autoapelează se numește *recursiv*.

De exemplu, presupunem că este definit tipul

```
type Natural = 0..MaxInt;
```

Funcția *factorial*

$$f(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n \cdot f(n-1), & \text{dacă } n > 0 \end{cases}$$

poate fi exprimată în PASCAL, urmînd direct definiția, în forma

```
function F(n : Natural) : Natural;
begin
 if n=0 then F:=1
 else F:=n*F(n-1)
end; { F}
```

Efectul unui apel  $F(7)$  este declanșarea unui lanț de apeluri ale funcției  $F$  pentru parametrii actuali 6, 5, ..., 2, 1, 0:

$$F(7) \rightarrow F(6) \rightarrow F(5) \rightarrow \dots \rightarrow F(1) \rightarrow F(0).$$

Apelul  $F(0)$  determină evaluarea directă a funcției și oprirea procesului repetitiv; urmează revenirile din apeluri și evaluarea lui  $F$  pentru 1, 2, ..., 6, 7, ultima valoare fiind întoarcerea în locul primului apel.

Funcția

$$\text{fib}(n) = \begin{cases} 0, & \text{dacă } n = 0; \\ 1, & \text{dacă } n = 1; \\ \text{fib}(n-1) + \text{fib}(n-2), & \text{dacă } n > 1 \end{cases}$$

are ca valori numerele lui Fibonacci. Urmînd definiția, obținem:

```
function Fib(n:Natural):Natural;
begin
 if n=0 then Fib:=0
 else if n=1 then Fib:=1
```

```

 else Fib:=Fib(n-1)+Fib(n-2)
end; { Fib}

```

Fiecare apel al funcției Fib pentru  $n > 1$  generează două apeluri Fib( $n-1$ ), Fib( $n-2$ ) ș.a.m.d., de exemplu:

```

Fib(4) -> Fib(3), Fib(2) -> Fib(2),
Fib(1), Fib(1), Fib(0) -> Fib(1), Fib(0).

```

Din exemplele în studiu se observă că recursia este utilă pentru programarea unor calcule repetitive. Repetiția este asigurată prin execuția unui subprogram care conține un apel la el însuși: când execuția ajunge la acest apel, este declanșată o nouă execuție ș.a.m.d.

Evident, orice subprogram recursiv trebuie să includă condiții de oprire a procesului repetitiv. De exemplu, în cazul funcției *factorial* procesul repetitiv se oprește când  $n$  ia valoarea 0; în cazul funcției Fib procesul se oprește când  $n$  ia valoarea 0 sau 1.

La orice apel de subprogram în memoria calculatorului vor fi depuse următoarele informații:

- valorile curente ale parametrilor transmiși prin valoare;
- locațiile (adresele) parametrilor-variabilă;
- adresa de retur, adică adresa instrucțiunii ce urmează după apel.

Prin urmare, la apeluri recursive spațiul ocupat din memorie va crește rapid, riscând depășirea capacității de memorare a calculatorului. Astfel de cazuri pot fi evitate, înlocuind recursia prin iterație (instrucțiunile **for**, **while**, **repeat**). Pentru exemplificare prezentăm o formă nerecursivă a funcției *factorial*:

```

function F(n: Natural): Natural;
var i, p : Natural;
begin
 p:=1;
 for i:=1 to n do p:=p*i;
 F:=p;
end; { F}

```

Recursia este deosebit de utilă în cazurile în care elaborarea unor algoritmi nerecursivi este foarte complicată: translatarea programelor PASCAL în limbajul cod-mașină, grafica pe calculator, recunoașterea formelor ș.a.

## Întrebări și exerciții

- ❶ Cum se execută un subprogram recursiv? Ce informații se depun în memoria calculatorului la execuția unui apel recursiv?
- ❷ Care este diferența dintre recursie și iterație?
- ❸ Elaborați o formă nerecursivă a funcției lui Fibonacci.
- ❹ Scrieți un subprogram recursiv care:



- a) calculează suma  $S(n) = 1 + 3 + 5 + \dots + (2n - 1)$ ;
- b) calculează produsul  $P(n) = 1 \times 4 \times 7 \times \dots \times (3n - 2)$ ;
- c) inversează un șir de caractere;
- d) calculează produsul  $P(n) = 2 \times 4 \times 6 \times \dots \times 2n$ .

⑤ Elaborați un program care citește de la tastatură numerele naturale  $m$ ,  $n$  și afișează pe ecran valoarea funcției lui Ackermann:

$$a(m, n) = \begin{cases} n + 1, & \text{dacă } m = 0 ; \\ a(m - 1, 1), & \text{dacă } n = 0 ; \\ a(m - 1, a(m, n - 1)), & \text{dacă } m > 0 \text{ și } n > 0. \end{cases}$$

Calculați  $a(0, 0)$ ,  $a(1, 2)$ ,  $a(2, 1)$  și  $a(2, 2)$ . Încercați să calculați  $a(4, 4)$  și  $a(10, 10)$ . Explicați mesajele afișate pe ecran.

⑥ Se consideră declarația

**type** Vector=**array** [ 1..20] **of** integer;

Elaborați un subprogram recursiv care:

- a) afișează componentele vectorului pe ecran;
- b) calculează suma componentelor;
- c) inversează componentele vectorului;
- d) calculează suma componentelor pozitive;
- e) verifică dacă cel puțin o componentă a vectorului este negativă;
- f) calculează produsul componentelor negative;
- g) verifică dacă cel puțin o componentă a vectorului este egală cu un număr dat.

⑦ Elaborați o formă nerecursivă a funcției ce urmează:

```
function S(n:Natural):Natural;
begin
 if n=0 then S:=0
 else S:=n+S(n-1)
end; { S}
```

⑧ Scrieți o funcție recursivă care returnează valoarea true dacă șirul de caractere  $s$  este conform definiției

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \mid \langle \text{Cifră} \rangle \langle \text{Număr} \rangle$

*Indicație.* Forma unei astfel de funcții derivă din formula metalingvistică. Varianta nerecursivă

```
function N(s : string) : boolean;
var i : integer;
 p : boolean
begin
 p:=(s<>'');
 for i=1 to length(s) do
 p:=p and (s[i] in ['0'..'9']);
 N:=p;
end;
```

derivă din definiția



$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

- ⑨ Se consideră următoarele formule metalingvistice:

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

$\langle \text{Semn} \rangle ::= + \mid -$

$\langle \text{Expresie} \rangle ::= \langle \text{Număr} \rangle \mid \langle \text{Expresie} \rangle \langle \text{Semn} \rangle \langle \text{Expresie} \rangle$

Scrieți o funcție recursivă care returnează valoarea `true` dacă șirul de caractere `s` este conform definiției unității lexicale  $\langle \text{Expresie} \rangle$ .

## 5.9. DECLARAȚII ANTICIPATE

Toate subprogramele recursive prezentate pînă acum includ în partea executabilă un autoapel, motiv pentru care se spune că s-a folosit *recursia directă*. Există însă situații în care un subprogram trebuie autoapelat prin intermediul altui subprogram, adică apare *recursia indirectă*.

*Exemplu:*

$$f(x) = \begin{cases} 1, & \text{dacă } x = 0; \\ x + g(x - 1), & \text{dacă } x > 0; \end{cases}$$

$$g(x) = \begin{cases} 2, & \text{dacă } x = 0; \\ x + f(x - 1), & \text{dacă } x > 0. \end{cases}$$

Evident, în textul PASCAL al funcției  $f(x)$  se va utiliza apelul  $g(x - 1)$ , iar în textul  $g(x)$  se va utiliza apelul  $f(x - 1)$ .

Conform regulilor de bază ale limbajului PASCAL, locul de definiție al unui nume trebuie să precede textual utilizările numelui respectiv. În cazul funcțiilor  $f(x)$  și  $g(x)$ , oricare ar fi forma textuală a programului, numele uneia dintre funcții apare înaintea locului de definire. Pentru a soluționa astfel de probleme, se utilizează declarațiile anticipate de subprograme.

O **declarație anticipată** include antetul subprogramului și directiva **forward** (înainte). Corpul subprogramului apare textual undeva mai jos și este precedat de un antet redus în care se specifică doar numele subprogramului, fără parametri și tipuri.

*Exemplu:*

```
Program P115;
{ Declarații anticipate }
type Natural=0..MaxInt;

function F(x : Natural) : Natural;
```

```

forward; { declarația anticipată a funcției F }

function G(x : Natural) : Natural;
 begin
 if x=0 then G:=2
 else G:=x+F(x-1);
 end; { G }

function F;
 begin { corpul funcției F }
 if x=0 then F:=1
 else F:=x+G(x-1);
 end; { F }

begin
 writeln(F(5));
 readln;
end.

```

În principiu, declarațiile anticipate pot fi folosite pentru definirea oricăror proceduri sau funcții, nu neapărat recursive. Menționăm însă că utilizarea nejustificată a declarațiilor anticipate complică structura de bloc a programelor PASCAL.

## Întrebări și exerciții

- ❶ Care este diferența dintre *recursia directă* și *recursia indirectă*?
- ❷ Este oare necesară utilizarea directivei **forward** în cazul recursiei indirecte? Argumentați răspunsul.
- ❸ Elaborați un program care citește de la tastatură numărul natural  $n$  și afișează pe ecran valoarea funcției  $f(n)$ :

$$f(n) = \begin{cases} 1, & \text{dacă } n=0; \\ g(n-1)+h(n-1), & \text{dacă } n>0; \end{cases}$$

$$g(n) = \begin{cases} 2, & \text{dacă } n=0; \\ f(n-1)+h(n-1), & \text{dacă } n>0; \end{cases}$$

$$h(n) = \begin{cases} 3, & \text{dacă } n=0; \\ f(n-1)+g(n-1), & \text{dacă } n>0. \end{cases}$$

Calculați valorile  $f(0)$ ,  $f(5)$ ,  $f(10)$  și  $f(15)$ . Încercați să calculați  $f(100)$  și  $f(200)$ . Explicați mesajele afișate pe ecran.

- ❹ Precizați ce va afișa pe ecran programul ce urmează:

```

Program P116;
 var x : real;

function F(x : real) : real; forward;

```

```

procedure P(var x : real); forward;
procedure Q(var x : real); forward;

function F;
 begin
 F:=2* x
 end;

procedure P;
begin
 x:=x+1
end;

procedure Q;
 begin
 x:=x+2
 end;

begin
 x:=1;
 P(x);
 Q(x);
 writeln(F(x));
 readln;
end.

```

## 5.10. SINTAXA DECLARAȚIILOR ȘI APELURILOR DE SUBPROGRAME

În general, definirea unei funcții se face cu ajutorul următoarelor formule metalingvistice:

$$\begin{aligned}
 \langle \text{Funcție} \rangle &::= \langle \text{Antet funcție} \rangle ; \langle \text{Corp} \rangle \mid \langle \text{Antet funcție} \rangle ; \\
 &\quad \langle \text{Directivă} \rangle \mid \mathbf{function} \langle \text{Identificator} \rangle ; \langle \text{Corp} \rangle \\
 \langle \text{Antet funcție} \rangle &::= \mathbf{function} \langle \text{Identificator} \rangle \\
 &\quad [\langle \text{Listă parametri formali} \rangle] : \langle \text{Identificator} \rangle
 \end{aligned}$$

Diagramele sintactice sînt prezentate în fig. 5.3.

Procedurile se definesc cu ajutorul următoarelor formule:

$$\begin{aligned}
 \langle \text{Procedură} \rangle &::= \langle \text{Antet procedură} \rangle ; \langle \text{Corp} \rangle \mid \langle \text{Antet procedură} \rangle ; \\
 &\quad \langle \text{Directivă} \rangle \mid \mathbf{procedure} \langle \text{Identificator} \rangle ; \langle \text{Corp} \rangle \\
 \langle \text{Antet procedură} \rangle &:= \mathbf{procedure} \langle \text{Identificator} \rangle \\
 &\quad [\langle \text{Listă parametri formali} \rangle]
 \end{aligned}$$

Diagramele sintactice sînt prezentate în fig. 5.4.



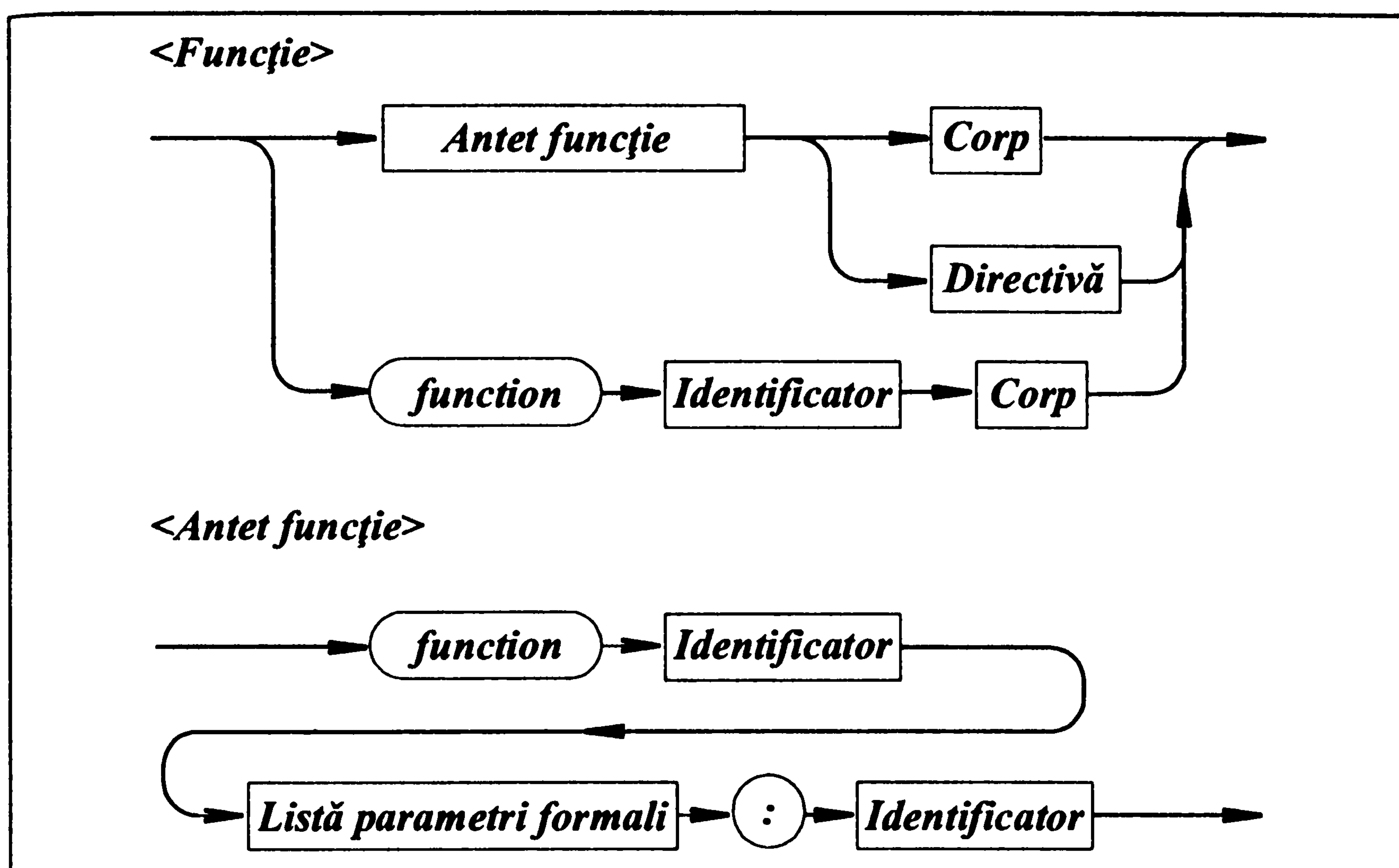


Fig. 5.3. Sintaxa declarațiilor de funcții

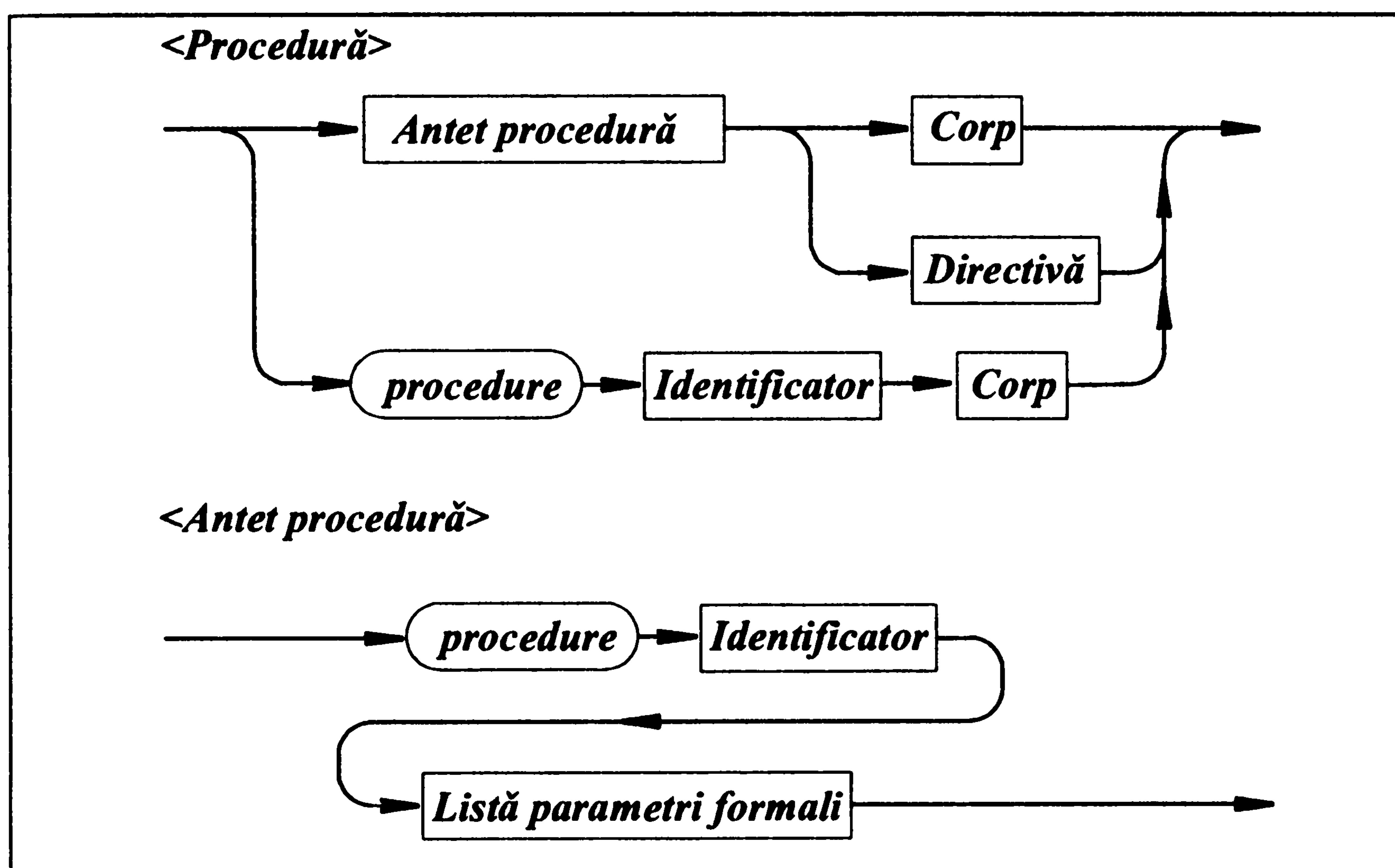


Fig. 5.4. Sintaxa declarațiilor de proceduri

**Listele de parametri formali** au următoarea sintaxă:

**<Listă parametri formali> ::=**  
                   (**<Parametru formal> { ; <Parametru formal> }**)

$$\begin{aligned} \langle \text{Parametru formal} \rangle &::= [\mathbf{var}] \langle \text{Identificator} \rangle \{ , \langle \text{Identificator} \rangle \} \\ &: \langle \text{Identificator} \rangle \mid \langle \text{Antet funcție} \rangle \mid \\ &\langle \text{Antet procedură} \rangle \end{aligned}$$

Diagrama sintactică este prezentată în fig. 5.5.

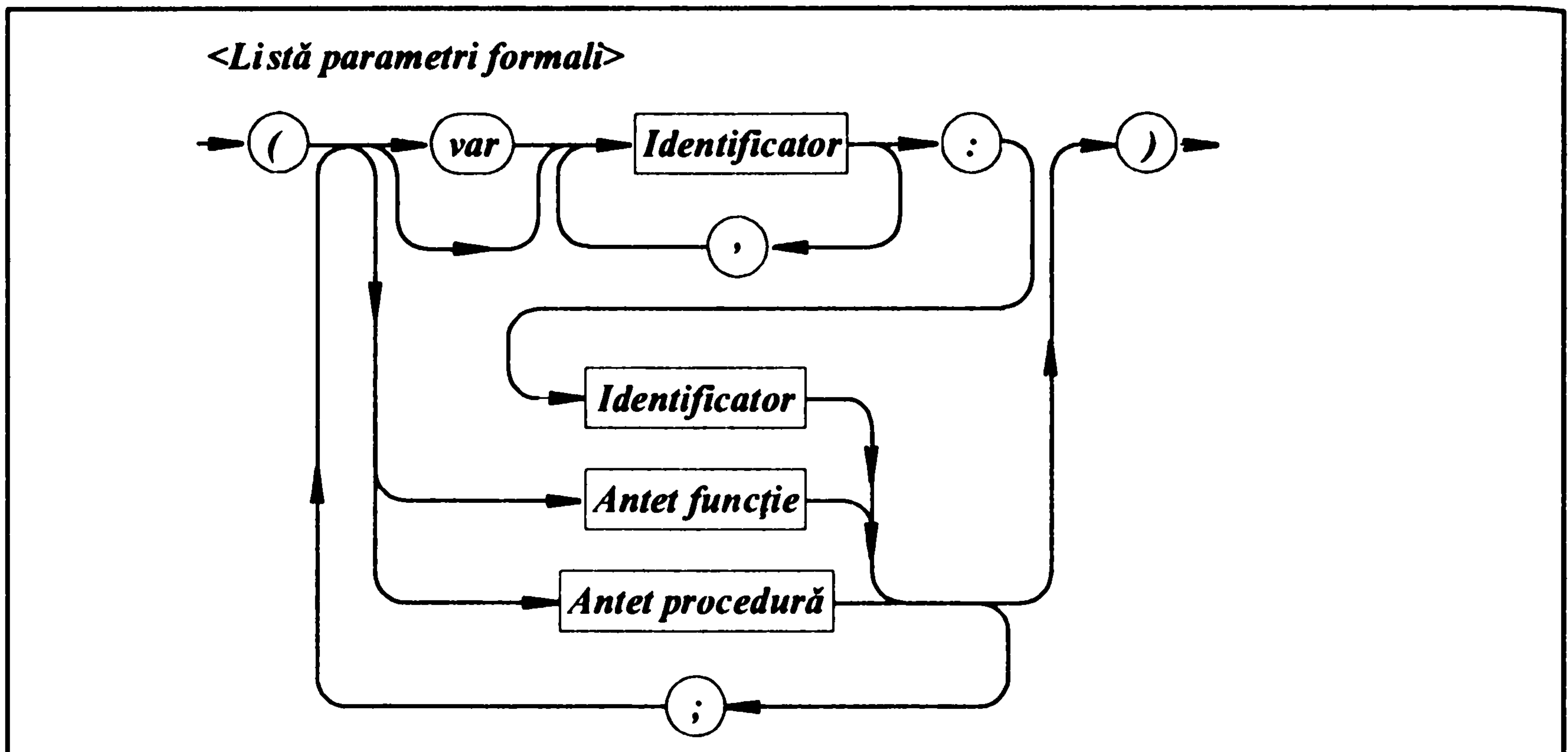


Fig. 5.5. Diagrama sintactică <Lista parametri formali>

Amintim că în lipsa cuvântului-cheie **var** identificatorii din listă specifică **parametrii-valoare**. Cuvântul **var** prefixează **parametrii-variabilă**. Antetul unei funcții (proceduri) din listă specifică un **parametru-funcție (procedură)**. În TurboPASCAL astfel de parametri se declară explicit ca aparținând unui **tip procedural** și au forma parametrilor-valoare. Limbajul PASCAL extinde sensul uzual al noțiunii de funcție, permițând returnarea valorilor nu numai prin numele funcției, ci și prin parametri-variabilă.

Un **apel de funcție** are forma:

$$\langle \text{Apel funcție} \rangle ::= \langle \text{Nume funcție} \rangle [\langle \text{Listă parametri actuali} \rangle]$$

iar o instrucțiune **apel de procedură**:

$$\langle \text{Apel procedură} \rangle ::= \langle \text{Nume procedură} \rangle [\langle \text{Listă parametri actuali} \rangle]$$

**Parametrii actuali** se specifică cu ajutorul formulelor:

$$\langle \text{Listă parametri actuali} \rangle ::=$$

$$(\langle \text{Parametru actual} \rangle \{ , \langle \text{Parametru actual} \rangle \})$$

$$\langle \text{Parametru actual} \rangle ::= \langle \text{Expresie} \rangle \mid \langle \text{Variabilă} \rangle \mid$$

$$\langle \text{Nume funcție} \rangle \mid \langle \text{Nume procedură} \rangle$$

Diagrama sintactică este prezentată în fig. 5.6.

Correspondența între un parametru actual și parametrul formal se face prin poziția ocupată de aceștia în cele două liste.

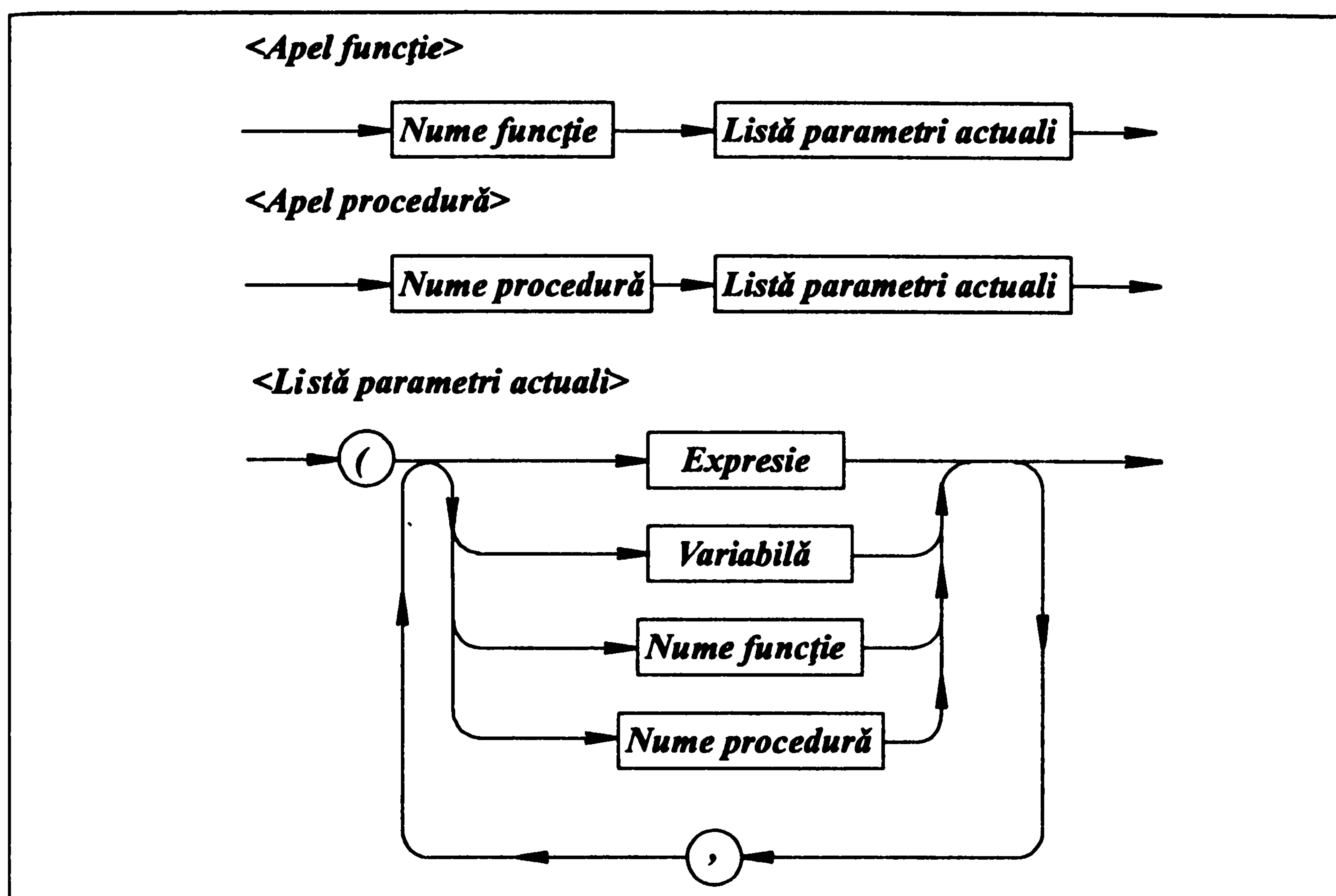


Fig. 5.6. Sintaxa apelurilor de funcții și proceduri

În cazul unui **parametru-valoare** drept parametru actual poate fi utilizată orice expresie, în particular, o constantă sau o variabilă. Expresia respectivă trebuie să fie compatibilă din punct de vedere al atribuirii cu tipul parametrului formal. Modificările parametrilor-valoare nu se transmit în exteriorul subprogramului.

În cazul unui **parametru-variabilă** drept parametri actuali pot fi utilizate numai variabile. Modificările parametrilor-variabilă se transmit în exteriorul subprogramului.

În cazul unui **parametru-funcție (procedură)** drept parametru actual poate fi utilizat orice nume de funcție (procedură) antetul căreia are forma specificată în lista parametrilor formali.

## Întrebări și exerciții

- ❶ Când se utilizează declarațiile de forma  
**function** <Identificator> ;<Corp> ?
- ❷ Indicați pe diagramele sintactice din fig. 5.3 și 5.5 drumurile care corespund declarațiilor de funcții din programul P115, paragraful 5.9.
- ❸ Care este diferența dintre un parametru-valoare și un parametru-variabilă?
- ❹ Indicați pe diagramele sintactice din fig. 5.4 și 5.5 drumurile care corespund declarațiilor de proceduri din programul P116, paragraful 5.9.
- ❺ Cum se specifică parametrii formali funcție (procedură) în PASCAL? În Turbo PASCAL?
- ❻ Indicați pe diagramele sintactice din fig. 5.3–5.6 drumurile care corespund declarațiilor și apelurilor de subprograme din programul P102, paragraful 5.4.



## Capitolul 6

.....

# STRUCTURI DINAMICE DE DATE

### 6.1. VARIABLE DINAMICE. TIPUL REFERINȚĂ

Variabilele declarate în secțiunea **var** a unui program sau subprogram se numesc **variabile statice**. Numărul variabilelor statice se stabilește în momentul scrierii programului și nu poate fi schimbat în timpul execuției. Există însă situații în care numărul necesar de variabile nu este cunoscut din timp.

De exemplu, presupunem că este necesară prelucrarea datelor referitoare la persoanele care formează un șir de așteptare (o coadă) la o casă de bilete. Lungimea cozii este nedefinită. De fiecare dată cum apare o persoană nouă, trebuie să se creeze o variabilă de tipul respectiv. După ce persoana pleacă, variabila corespunzătoare devine inutilă.

Variabilele care sînt create și eventual distruse în timpul execuției programului se numesc **variabile dinamice**.

Accesul la variabilele dinamice se face prin intermediul variabilelor de tip *referință*. De obicei, un tip *referință* se definește printr-o declarație de forma:

**type**  $T_r = ^T_b$ ;

unde  $T_r$  este numele tipului *referință*, iar  $T_b$  este tipul de bază. Semnul “^” se citește “adresă”. Evident, pot fi utilizate și tipuri referință anonime. Diagrama sintactică <Tip referință> este prezentată în fig. 6.1.

Mulțimea de valori ale unui tip de date *referință* constă din adrese.

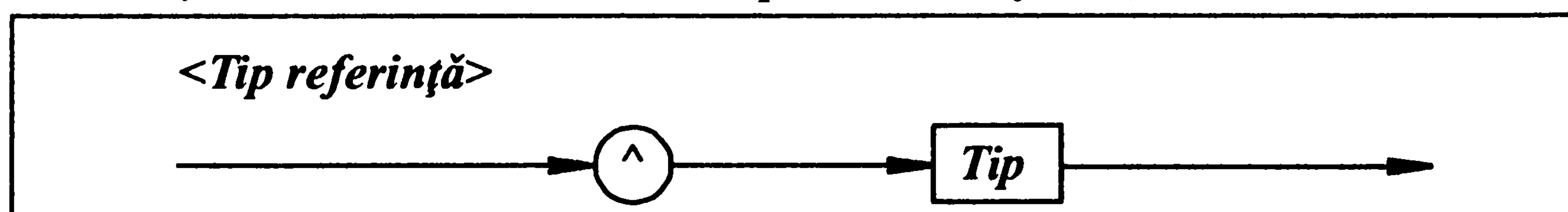


Fig. 6.1. Diagrama sintactică <Tip referință>

Fiecare adresă identifică o variabilă dinamică ce aparține tipului de bază. La această mulțime de adrese se mai adaugă o valoare specială, notată **nil** (zero) care nu identifică nici o variabilă.

*Exemplu:*

```
type AdresaInteger=^integer;
 AdresaChar=^char;
var i : AdresaInteger;
 r : ^real;
 c : AdresaChar;
```

Valoarea curentă a variabilei *i* va indica o variabilă dinamică de tipul *integer*. Într-un mod similar, variabilele de tip referință *r* și *c* identifică variabile de tipul *real* și, respectiv, *char*. Subliniem faptul că tipurile de date *AdresaInteger*, *AdresaChar* și tipul anonim <sup>^</sup>*real* sînt tipuri *referință* distincte.

Operațiile care se pot face cu valorile unui tip de date *referință* sînt = și <>. Valorile de tip *referință* nu pot fi citite de la tastatură și afișate pe ecran.

**Crearea** unei variabile dinamice se realizează cu procedura predefinită *new* (nou). Apelul acestei proceduri are forma

*new* (*p*)

unde *p* este o variabilă de tip referință.

Procedura alocă spațiu de memorie pentru variabila nou creată și returnează adresa zonei respective prin variabila *p*. În continuare variabila dinamică poate fi accesată prin așa-zisa **dereperare**: numele variabilei de tip *referință* *p* este urmat de semnul de “^”. Dereperarea unei variabile de tip *referință* cu conținutul *nil* va declanșa o eroare de execuție.

*Exemplu:*

*new* (*i*) ; *i*<sup>^</sup>:=1 — crearea unei variabile dinamice de tipul *integer*; variabilei create *i* se atribuie valoarea 1;

*new* (*r*) ; *r*<sup>^</sup>:=2.0 — crearea unei variabile dinamice de tipul *real*; variabilei create *r* se atribuie valoarea 2.0;

*new* (*c*) ; *c*<sup>^</sup>:= '\*' — crearea unei variabile dinamice de tipul *char*; variabilei create *c* se atribuie valoarea '\*'.

Subliniem faptul că variabila dinamică *p*<sup>^</sup> obținută printr-un apel *new* (*p*) este distinctă de toate variabilele create anterior. Prin urmare, executarea instrucțiunilor

*new* (*p*) ; *new* (*p*) ; ... ; *new* (*p*)

conduce la crearea unui șir  $v_1, v_2, \dots, v_n$  de variabile dinamice. Numai ultima variabilă creată,  $v_n$ , este referită prin *p*<sup>^</sup>. Întrucît valorile variabilelor de tip *referință* reprezintă adresele anumitor zone din memoria internă a calculatorului, variabilele în studiu se numesc **indicatori de adresă**.



**Distrugerea** unei variabile dinamice și eliberarea zonei respective de memorie se realizează cu procedura predefinită `dispose` (a dispune). Apelul acestei proceduri are forma:

`dispose (p)`

unde `p` este o variabilă de tip *referință*.

*Exemple:*

`dispose (i); dispose (r); dispose (c)`

După executarea instrucțiunii `dispose (p)` valoarea variabilei de tip *referință* `p` este nedefinită.

Asupra variabilelor dinamice se pot efectua toate operațiile admise de tipul de bază.

*Exemplu:*

```
Program P117;
{Operații cu variabile dinamice }
type AdresaInteger=^integer;
var i, j, k : AdresaInteger;
 r, s, t : ^real;
begin
 { crearea variabilelor dinamice de tipul integer }
 new(i); new(j); new(k);
 { operații cu variabilele create }
 i^:=1; j^:=2;
 k^:=i^+j^;
 writeln(k^);
 { crearea variabilelor dinamice de tipul real }
 new(r); new(s); new(t);
 { operații cu variabilele create }
 r^:=1.0; s^:=2.0;
 t^:=r^/s^;
 writeln(t^);
 { distrugerea variabilelor dinamice }
 dispose(i); dispose(j); dispose(k);
 dispose(r); dispose(s); dispose(t);
 readln;
end.
```

Spre deosebire de variabilele statice, care ocupă zone de memorie stabilite de compilator, variabilele dinamice ocupă zone de memorie oferite de procedura `new`. Zonele respective sînt eliberate de procedura `dispose` și pot fi reutilizate pentru crearea unor variabile dinamice noi. Prin urmare, procedurile `new` și `dispose` asigură **alocarea (rezervarea) dinamică a memoriei**: spațiul de memorie este atribuit unei variabile dinamice numai pe durata existenței ei.



Numărul de variabile dinamice ce pot exista concomitent în timpul execuției unui program PASCAL depinde de tipul variabilelor și spațiul de memorie disponibil. În cazul în care tot spațiul de memorie este deja ocupat, apelul procedurii new va declanșa o eroare de execuție.

*Exemplu:*

```
Program P118;
{ Eroare: depășirea capacității memoriei }
label 1;
var i : ^integer;
begin
 1 : new(i);
 goto 1;
end.
```

Alocarea dinamică a memoriei necesită o atenție sporită din partea programatorului care este obligat să asigure crearea, distrugerea și referirea corectă a variabilelor dinamice.

### Întrebări și exerciții

- ❶ Care este diferența dintre variabilele statice și variabilele dinamice?
- ❷ Cum se identifică variabilele dinamice?
- ❸ Indicați pe diagrama sintactică din *fig. 6.1* drumurile care corespund declarațiilor de tipuri referință din programul P117.
- ❹ Se consideră declarațiile:

```
type AdresaReal = ^real;
var r : AdresaReal;
```

Precizați mulțimea de valori ale tipului de date AdresaReal și mulțimea de valori pe care le poate lua variabila dinamică r^.

- ❺ Ce operații se pot efectua cu valorile unui tip de date *referință*? Cu variabilele dinamice?
- ❻ Se consideră declarațiile:

```
type AdresaTablou = ^array[1..10] of integer;
var t : AdresaTablou.
```

Precizați mulțimea de valori ale tipului de date AdresaTablou și mulțimea de valori pe care le poate lua variabila dinamică t^.

- ❼ Comentați programul:

```
Program P119;
{ Eroare }
var r, s : ^real;
begin
 r^:=1; s^:=2;
 writeln('r^=', r^, ' s^=', s^);
 readln;
end.
```

- ⑧ Elaborați un program în care se creează două variabile dinamice de tipul *șir de caractere*. Atribuiți valori variabilelor create și afișați la ecran rezultatul concatenării șirurilor respective.
- ⑨ Ce va afișa pe ecran programul ce urmează?

```
Program P120;
var i : ^integer;
begin
 new(i); i^:=1;
 new(i); i^:=2;
 new(i); i^:=3;
 writeln(i^);
 readln;
end.
```

- ⑩ Comentați programul:

```
Program P121;
{ Eroare }
var i, j : ^integer;
begin
 new(i);
 i^:=1;
 dispose(i);
 new(j);
 j^:=2;
 dispose(j);
 writeln('i^=', i^, ' j^=', j^);
 readln;
end.
```

- ⑪ Explicați expresia *alocarea dinamică a memoriei*.

## 6.2. STRUCTURI DE DATE

O **structură de date** este formată din datele propriu-zise și relațiile dintre ele. În funcție de modul de organizare, o structură de date poate fi implicită sau explicită.

Tablourile, șirurile de caractere, articolele, fișierele și mulțimile studiate în capitolele precedente sînt **structuri implicite de date**. Relațiile dintre componentele acestor structuri sînt predefinite și nemodificabile. De exemplu, toate componentele unui șir de caractere au un nume comun, iar caracterul  $s[i+1]$  este succesorul caracterului  $s[i]$  în virtutea poziției ocupate.

Întrucît structura tablourilor, șirurilor de caractere, articolelor, mulțimilor și fișierelor nu se modifică în timpul execuției oricărui pro-



gram sau subprogram, variabilele respective reprezintă **structuri statice de date**.

Folosind date cu structură implicită, putem rezolva reprezentativ o clasă limitată de probleme. În multe cazuri relațiile dintre componente nu numai că se modifică dinamic, dar în același timp, pot deveni deosebit de complexe.

De exemplu, în cazul unui fir de așteptare la o casă de bilete relațiile dintre persoane se modifică: persoanele nou-sosite se așază la rând; persoanele în criză de timp pleacă fără să-și mai procure bilete; persoanele care au plecat pentru un timp își păstrează rândul ș.a.m.d. În cazul proiectării asistate de calculator a rețelelor de circulație, stațiile, rutele, capacitatea de trafic ș.a. pot fi stabilite interactiv de către utilizator. În astfel de situații utilizarea datelor cu structură implicită devine nenaturală, dificilă și inefficientă.

Prin urmare, este necesară folosirea unor structuri de date în care relațiile dintre componente să fie reprezentate și prelucrate în mod explicit. Acest efect se poate obține atașând fiecărei componente o informație ce caracterizează relațiile acesteia cu alte date ale structurii. În cele mai multe cazuri, informația suplimentară, numită **informație de structură**, se reprezintă prin variabilele de tipul referință.

Structurile de date componentele cărora sînt create și eventual distruse în timpul execuției programului se numesc **structuri dinamice de date**. Structurile dinamice frecvent utilizate sînt: listele unidirecționale, listele bidirecționale, stivele, cozile, arborii ș. a.

O structură de date este **recursivă** dacă ea poate fi descompusă în date cu aceeași structură. Pentru exemplificare menționăm listele unidirecționale și arborii care vor fi studiați în paragrafele următoare.

## Întrebări și exerciții

- ❶ Explicați termenul *structură de date*. Dați exemple.
- ❷ Care este diferența dintre structurile implicite și structurile explicite de date?
- ❸ O structură de date este **omogenă** dacă toate componentele sînt de același tip. În caz contrar structura de date este **eterogenă**. Dați exemple de structuri omogene și structuri eterogene de date.
- ❹ Care este diferența dintre structurile statice și structurile dinamice de date?
- ❺ Explicați termenul *structură recursivă de date*.

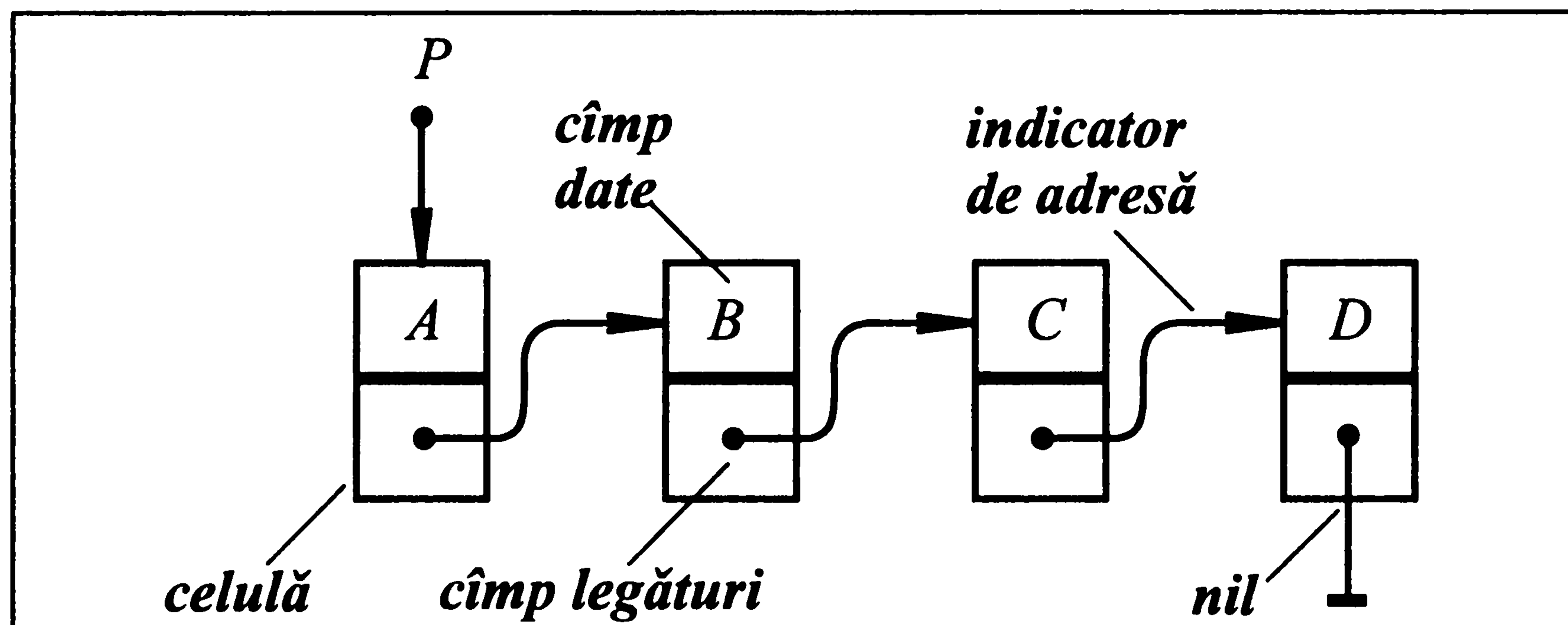
## 6.3. LISTE UNIDIRECȚIONALE

Listele **unidirecționale** sînt structuri explicite și dinamice de date formate din celule. Fiecare **celulă** este o variabilă dinamică de tipul **record** ce conține, în principal, două cîmpuri: cîmpul datelor și



cîmpul legăturilor. **Cîmpul datelor** memorează informația prelucrabilă asociată celulei. **Cîmpul legăturilor** furnizează indicatorul de adresă corespunzător celulei la care se poate ajunge din celula curentă. Se consideră că orice celulă poate fi atinsă pornind de la o celulă privilegiată, numită **baza listei**.

Pentru exemplificare, în *fig. 6.2* este prezentată o listă unidirecțională formată din 4 celule. Celulele conțin elementele A, B, C și D.



*Fig. 6.2.* Listă unidirecțională

Datele necesare pentru crearea și prelucrarea unei liste unidirecționale pot fi definite prin declarații de forma:

```
type AdresaCelula = ^Celula;
 Celula = record
 Info : string;
 Urm : AdresaCelula;
 end;
var P : AdresaCelula;
```

Informația utilă asociată unei celule se memorează în cîmpul Info, iar adresa celulei următoare în cîmpul Urm. Pentru simplificare se consideră că cîmpul Info este de tipul șir de caractere. Ultima celulă din listă va avea în cîmpul Urm valoarea **nil**. Adresa primei celule (adresa de bază) este memorată în variabila de tip referință P (*fig. 6.2*).

Subliniem faptul că în definiția tipului referință AdresaCelula tipul de bază Celula încă nu este cunoscut. Conform regulilor limbajului PASCAL, acest lucru este posibil numai în cazul variabilelor dinamice, cu condiția ca tipul de bază să fie definit mai târziu în aceeași declarație.

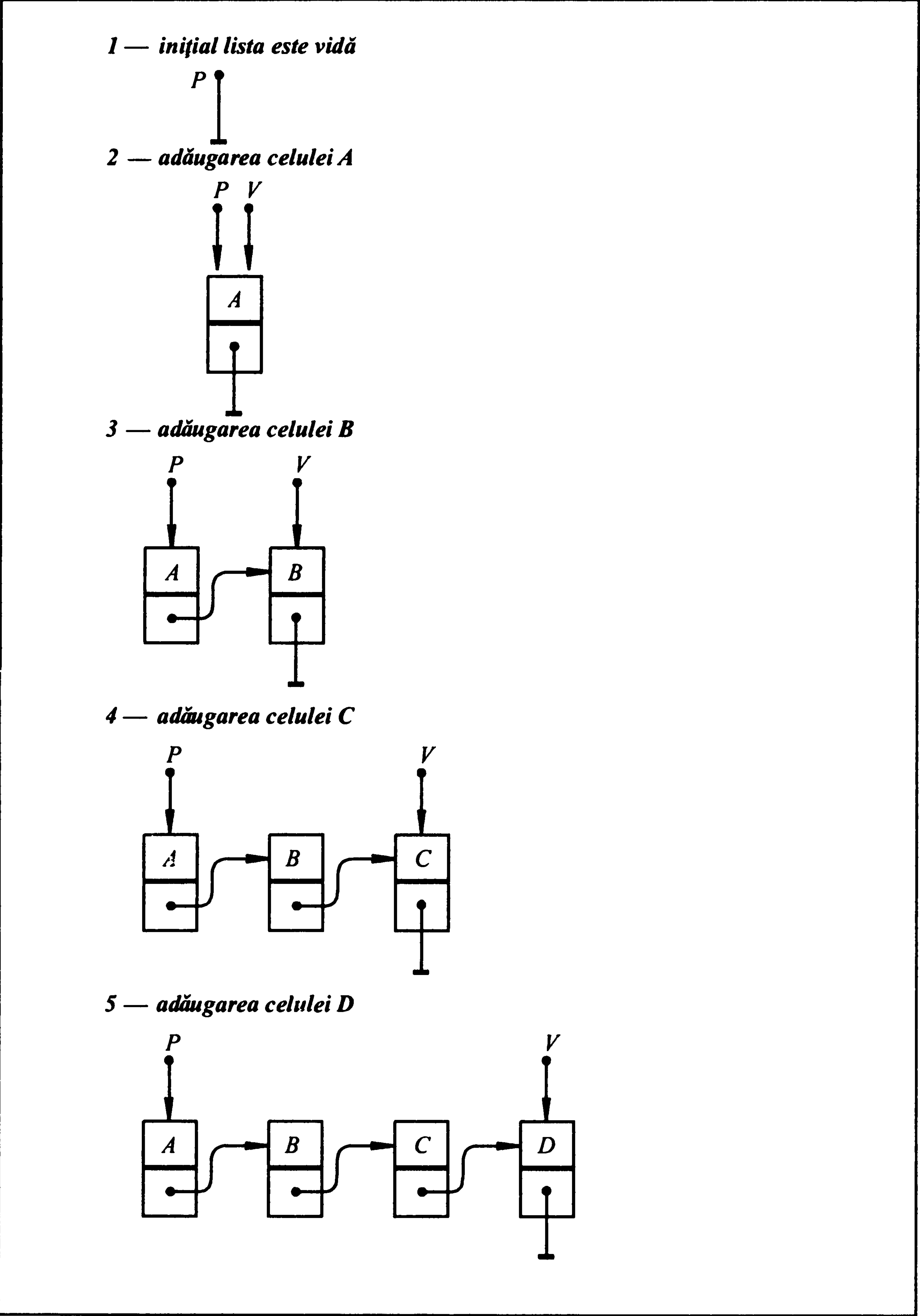
O listă unidirecțională poate fi creată adăugînd la vîrfurile listei cîte un element. Inițial lista în curs de construcție este vidă, adică nu conține nici o celulă.

*Exemplu:*

```
Program P122;
{ Crearea listei unidirecționale A->B->C->D }
type AdresaCelula = ^Celula;
 Celula = record
 Info : string;
 Urm : AdresaCelula;
 end;
var P, { adresa de bază }
 R, V : AdresaCelula;
begin
 { 1 - inițial lista este vidă }
 P := nil;
 { 2 - adăugarea celulei A }
 new(R); { crearea unei celule }
 P := R; { inițializarea adresei de bază }
 R^.Info := 'A'; { încărcarea informației utile }
 R^.Urm := nil; { înscrierea indicatorului
 "sfârșit de listă" }
 V := R; { memorarea adresei vârfului }
 { 3 - adăugarea celulei B }
 new(R); { crearea unei celule }
 R^.Info := 'B'; { încărcarea informației utile }
 R^.Urm := nil; { înscrierea indicatorului
 "sfârșit de listă" }
 V^.Urm := R; { crearea legăturii A -> B }
 V := R; { actualizarea adresei vârfului }
 { 4 - adăugarea celulei C }
 new(R); { crearea unei celule }
 R^.Info := 'C'; { încărcarea informației utile }
 R^.Urm := nil; { înscrierea indicatorului
 "sfârșit de listă" }
 V^.Urm := R; { crearea legăturii B -> C }
 V := R; { actualizarea adresei vârfului }
 { 5 - adăugarea celulei D }
 new(R); { crearea unei celule }
 R^.Info := 'D'; { încărcarea informației utile }
 R^.Urm := nil; { înscrierea indicatorului
 "sfârșit de listă" }
 V^.Urm := R; { crearea legăturii C -> D }
 V := R; { actualizarea adresei vârfului }
 { afișarea listei create }
 R := P;
 while R <> nil do begin
 writeln(R^.Info);
 R := R^.Urm
 end;

 readln;
end.
```

Procesul de construire a listei în studiu este prezentat în *fig. 6.3*. Variabila *V* (adresa vârfului) din programul P122 reține adresa ultimei celule deja create pentru a-i poziționa indicatorul de adresă



*Fig. 6.3.* Crearea listei unidirecționale



V<sup>^</sup>.Urm. Se procedează astfel pentru că în momentul în care completăm câmpurile R<sup>^</sup>.Info și R<sup>^</sup>.Urm ale celulei curente încă nu se cunoaște adresa celulei ce urmează.

Listele cu un număr arbitrar de celule pot fi create și afișate cu ajutorul procedurilor respective din programul P123:

```
Program P123;
{ Crearea listelor unidirectionale}
type AdresaCelula=^Celula;
 Celula=record
 Info : string;
 Urm : AdresaCelula;
 end;
var p,q,r : AdresaCelula;
 s : string;
 i: integer;
procedure Creare;
begin
 p:=nil;
 write('s='); readln(s);
 new(r); r^.Info:=s; r^.Urm:=nil;
 p:=r; q:=r;
 write('s=');
 while not eof do
 begin
 readln(s);write('s=');
 new(r); r^.Info:=s; r^.Urm:=nil;
 q^.Urm:=r; q:=r
 end;
end; { Creare }
procedure Afișare;
begin
 r:=p;
 while r<>nil do
 begin
 writeln(r^.Info);
 r:=r^.Urm;
 end;
 readln;
end; { Afișare }

begin
 Creare;
 Afișare;
end.
```

Orice listă unidirecțională poate fi definită **recursiv** după cum urmează:

a) o celulă este o listă unidirecțională;

b) o celulă ce conține o legătură către o altă listă unidirecțională este o listă unidirecțională.

Pentru a sublinia faptul că listele unidirecționale sînt structuri recursive de date, declarațiile respective pot fi transcrise în forma:

```
type Lista=^Celula;
 Celula=record
 Info : string;
 Urm : Lista
 end;
var P : Lista;
```

Proprietățile listelor unidirecționale pot fi reproduse parțial prin memorarea elementelor respective într-un tablou unidimensional. De exemplu, datele din *fig. 6.2* pot fi reprezentate în forma:

```
var L : array [1..4] of string;
 . . .
 L[1] := ' A ' ;
 L[2] := ' B ' ;
 L[3] := ' C ' ;
 L[4] := ' D ' ;
 . . .
```

Însă o astfel de reprezentare nu permite crearea și prelucrarea structurilor de date cu un număr arbitrar de elemente.

## Întrebări și exerciții

- ❶ Cum se definesc datele necesare pentru crearea unei liste?
- ❷ Care este destinația câmpului datelor din componența unei celule? Care este destinația câmpului legăturilor? Ce informație se înscrie în acest câmp?
- ❸ Scrieți un program care creează lista unidirecțională din *fig. 6.2*, adăugînd cîte un element la baza listei.
- ❹ Elaborați o procedură care schimbă cu locul două elemente din listă.
- ❺ De la tastatură se citesc numere întregi diferite de zero. Se cere să se creeze două liste, una a numerelor negative, iar alta — a numerelor pozitive.
- ❻ Prin ce se explică faptul că listele unidirecționale sînt structuri recursive de date?

## 6.4. PRELUCRAREA LISTELOR UNIDIRECȚIONALE

**Operațiile** frecvent utilizate în cazul listelor unidirecționale sînt:

- parcurgerea listei și prelucrarea informației utile asociate fiecărei celule;
- căutarea unui anumit element, identificat prin valoarea sa;
- includerea (inserarea) unui element într-un anumit loc din listă;
- excluderea (ștergerea) unui element dintr-o listă ș.a.

Presupunem că există o listă nevidă (fig. 6.2) definită prin declarațiile:

```
type AdresaCelula=^Celula;
 Celula=record;
 Info : string;
 Urm : AdresaCelula
 end;
```

Variabila P indică adresa de bază a listei în studiu.

**Parcurgerea** listei se realizează conform următoarei secvențe de instrucțiuni:

```
R:=P; { poziționare pe celula de bază }
while R<>nil do
begin
 { prelucrarea informației din câmpul R^.Info }
 R:=R^. Urm; { poziționare pe celula următoare }
end
```

**Căutarea** celulei ce conține elementul specificat de variabila Cheie este realizată de secvența:

```
R:=P;
while R^.Info<>Cheie do R:=R^.Urm
```

Adresa celulei în studiu va fi reținută în variabila R.

Subliniem faptul că această secvență se execută corect numai în cazul în care lista include cel puțin o celulă ce conține informația căutată. În caz contrar se ajunge la vârful listei, variabila R ia valoarea **nil**, iar dereperarea R^ provoacă o eroare de execuție. Pentru a evita astfel de erori, se utilizează secvența:

```
R:=P;
while R<>nil do
begin
 if R^.Info=Cheie then goto 1;
 R:=R^.Urm
end;
1:...
```

Întrucât listele unidirecționale sînt structuri recursive de date, operația de căutare poate fi realizată și de un subprogram recursiv:

```
type Lista=^AdresaCelula;
 Celula=record;
 Info : string;
 Urm : Lista;
 end;
var P : Lista;
...
function Caut(P : Lista; Cheie : string):Lista;
```



```

begin
 if P=nil then
 Caut:=nil
 else
 if P^.Info=Cheie then
 Caut:=P
 else Caut:=Caut(P^.Urm, Cheie)
 end;

```

Funcția Caut returnează adresa de bază a sublistei ce conține în prima celulă, dacă există, elementul specificat de parametrul Cheie.

**Includerea** celulei referite de indicatorul Q după celula referită de indicatorul R (fig. 6.4) este realizată de secvența de instrucțiuni:

```

Q^.Urm:=R^.urm;
R^.Urm:=Q

```

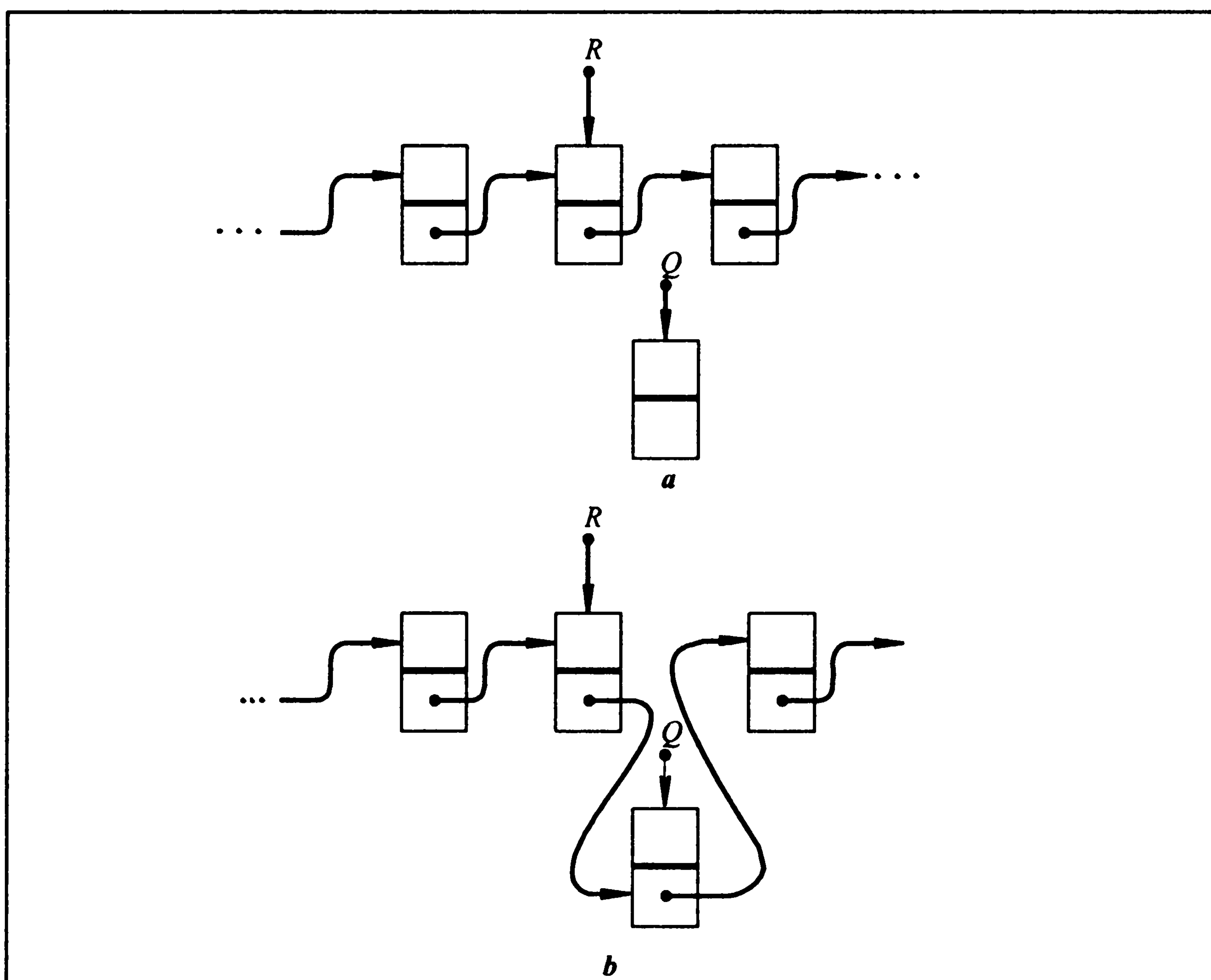


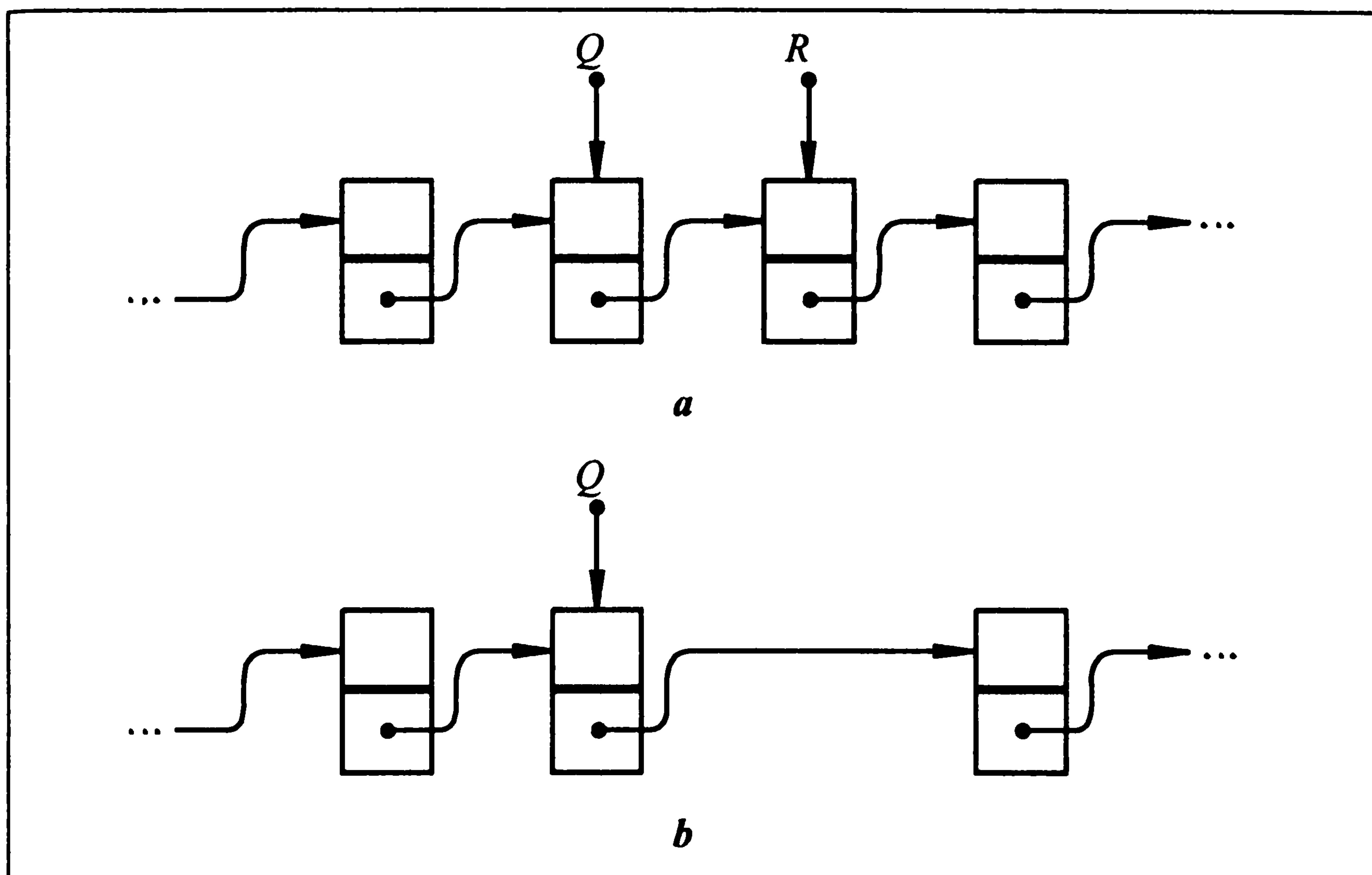
Fig. 6.4. Includerea unui element în listă:  
*a* – lista pînă la includere; *b* – lista după includere

**Excluderea** celulei R din listă necesită aflarea adresei Q a celulei precedente și modificarea indicatorului de adresă Q^.Urm (fig. 6.5):

```

Q:=P;
while Q^.Urm<>R do Q:=Q^.Urm;
Q^.Urm:=R^.Urm;

```



*Fig. 6.5. Excluderea unui element din listă:  
 a – lista pînă la excludere; b – lista după excludere*

Menționăm că includerea sau excluderea elementului din baza listei necesită actualizarea adresei de bază P.

*Exemplu:*

```

Program P124;
{ Crearea și prelucrarea unei liste unidirecționale }
type AdresaCelula = ^Celula;
 Celula = record
 Info : string;
 Urm : AdresaCelula;
 end;
var P : AdresaCelula; { adresa de bază }
 c : char;
procedure Creare;
var R, V : AdresaCelula;
begin
if P <> nil then writeln('Lista există deja')
else begin
 writeln('Dați lista:');
 while not eof do
 begin
 new(R);
 readln(R^.Info);
 R^.Urm := nil;
 if P = nil then begin P := R; V := R end
 else begin V^.Urm := R; V := R

```

```

 end;
 end;
 end;
end; { Creare }

procedure Afis;
var R : AdresaCelula;
begin
 if P=nil then writeln('Lista este vidă')
 else begin
 writeln('Lista curentă:');
 R:=P;
 while R<>nil do
 begin
 writeln(R^.Info);
 R:=R^.Urm;
 end;
 end;

 readln;
end; { Afis }
procedure Includ;
label 1;
var Q, R : AdresaCelula;
 Cheie : string;
begin
 new(Q);
 write('Dați elementul ce urmează');
 writeln(' să fie inclus:');
 readln(Q^.Info);
 write('Indicați elementul după care');
 writeln(' se va face includerea:');
 readln(Cheie);
 R:=P;
 while R<>nil do
 begin
 if R^.Info=Cheie then goto 1;
 R:=R^.Urm;
 end;
1:if R=nil then begin
 writeln('Element inexistent');
 dispose(Q);
 end;
 else begin
 Q^.Urm:=R^.Urm;
 R^.Urm:=Q;
 end;

end; { Includ }
procedure Exclud;
label 1;
var Q, R : AdresaCelula;

```



```

 Cheie : string;
begin
 write('Dați elementul ce urmează');
 writeln(' să fie exclus:');
 readln(Cheie);
 R:=P;
 Q:=R;
 while R<>nil do
 begin
 if R^.Info=Cheie then goto 1;
 Q:=R;
 R:=R^.Urm;
 end;
1:if R=nil then writeln('Element inexistent')
 else begin
 if R=P then P:=R^.Urm
 else Q^.Urm:=R^.Urm;
 dispose(R);
 end;
end; { Exclud }
begin
 P:=nil; { inițial lista este vidă }
 repeat
 writeln('Meniu:');
 writeln('C - Crearea listei');
 writeln('I - Includerea elementului');
 writeln('E - Excluderea elementului');
 writeln('A - Afișarea listei la ecran');
 writeln('O - Oprirea programului');
 write('Opțiunea='); readln(c);
 case c of
 'C' : Creare;
 'I' : Includ;
 'E' : Exclud;
 'A' : Afis;
 'O' :
 else writeln('Opțiune necunoscută')
 end;
 until c='O';
end.

```

Procedura Creare formează o listă unidirecțională cu un număr arbitrar de celule. Informația utilă asociată fiecărei celule se citește de la tastatură. Procedura Afis afișează elementele listei la ecran. Procedura Includ citește de la tastatură elementul ce urmează să fie inclus și elementul după care se va face includerea. În continuare se caută celula ce conține elementul specificat, după care, dacă există, este inclusă celula nou creată. Procedura Exclud caută și elimină, dacă există, celula ce conține elementul citit de la tastatură.

## Întrebări și exerciții

- ❶ Scrieți o funcție nerecursivă care returnează adresa vârfului listei unidirecționale. Transcrieți această funcție într-o formă recursivă.
- ❷ Transcrieți procedurile `Includ` și `Exclud` din programul P124, fără a utiliza instrucțiunea `goto`.
- ❸ Se consideră următoarele tipuri de date:

```
type AdresaCandidat = ^Candidat;
 Candidat = record
 NumePrenume : string;
 NotaMedie : real;
 Urm : AdresaCandidat
 end;
```

Elaborați un program care:

- a) creează o listă unidirecțională cu componente de tipul `Candidat`;
  - b) afișează lista pe ecran;
  - c) exclude din listă candidatul care își retrage actele;
  - d) include în listă candidatul care depune actele;
  - e) afișează pe ecran candidații cu media mai mare de 7,5;
  - f) creează o listă suplimentară formată din candidații cu media mai mare de 9,0;
  - g) exclude din listă toți candidații cu media mai mică de 6,0.
- ❹ Elaborați o procedură care:
- a) reordonează elementele listei unidirecționale conform unui anumit criteriu;
  - b) concatenează două liste unidirecționale;
  - c) descompune o listă în două liste;
  - d) selectează din listă elementele care corespund unui anumit criteriu.
- ❺ Elementele listei unidirecționale sînt memorate într-un tablou unidirecțional. Elaborați procedurile necesare pentru:
- a) parcurgerea listei;
  - b) căutarea unui anumit element;
  - c) includerea unui element;
  - d) excluderea unui element.

Care sînt avantajele și neajunsurile acestei reprezentări? Se consideră că lista va conține cel mult 100 de elemente.

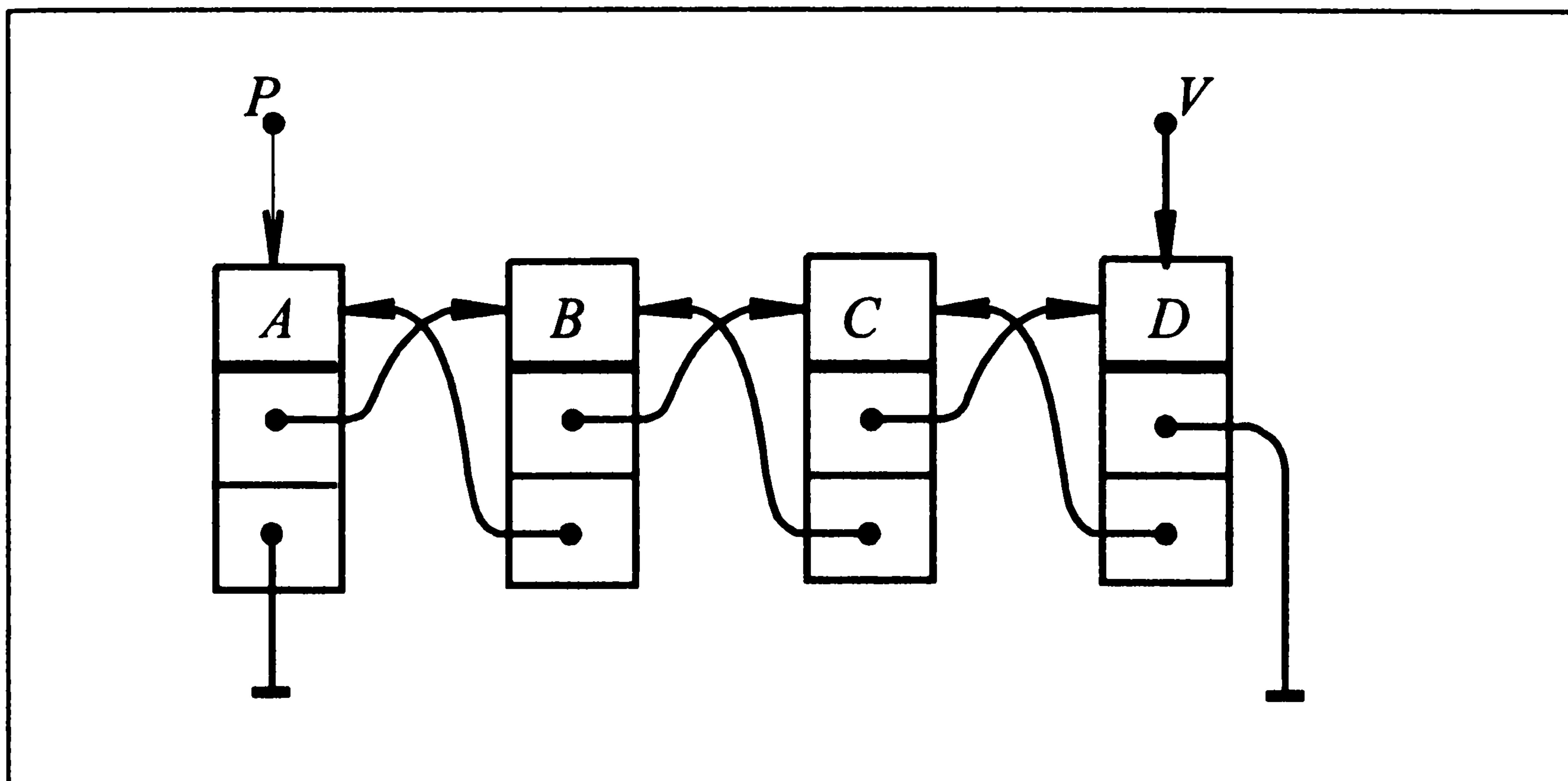
- ❻ Scrieți o funcție recursivă ce returnează numărul de celule dintr-o listă unidirecțională.
- ❼ Scrieți un subprogram recursiv care exclude o anumită celulă din listă.
- ❽ Prin **cuvînt** se înțelege orice secvență nevidă formată din literele alfabetului latin. Elaborați un program care formează lista cuvintelor întîlnite într-un fișier *text* și calculează numărul de apariții al fiecărui cuvînt. Examinați cazurile:
- a) cuvintele se includ în listă în ordinea primei lor apariții în text;
- b) cuvintele se includ în listă în ordine alfabetică.

Se consideră că literele mari și mici sînt identice.



## 6.5. LISTE BIDIRECȚIONALE

Fiecare celulă a unei liste bidirecționale conține în câmpul legăturilor doi indicatori de adresă. Primul indicator identifică celula următoare, iar al doilea — celula precedentă. Pentru exemplificare, în *fig. 6.6* este prezentată o listă bidirecțională formată din 4 celule ce conțin elementele A, B, C și D.



*Fig. 6.6.* Listă bidirecțională

Datele necesare pentru crearea și prelucrarea unei liste bidirecționale pot fi definite prin declarații de forma:

```
type AdresaCelula=^Celula;
 celula=record
 Info : string;
 Urm, Prec : AdresaCelula;
 end;
var P, V : AdresaCelula;
```

Variabilele de tip *referință* P și V memorează respectiv adresa primei celule (adresa de bază) și adresa ultimei celule (adresa vârfului).

O listă bidirecțională poate fi creată, adăugând la vârful listei câte un element. Inițial lista în curs de construcție este vidă.

*Exemplu:*

```
Program P125;
{ Crearea listei bidirecționale A<->B<->C<->D }
type AdresaCelula=^Celula;
 Celula=record
 Info : string;
 Urm, Prec : AdresaCelula;
 end;
```



```

var P, { adresa de bază }
 V, { adresa vârfului }
 R : AdresaCelula;
begin
 { 1 - inițial lista este vidă }
 P:=nil; V:=nil;
 { 2 - adăugarea celulei A }
 new(R);
 P:=R; { inițializarea adresei de bază }
 R^.Info:='A';
 R^.Prec:=nil;
 R^.Urm:=nil;
 V:=R; { inițializarea adresei vârfului }
 { 3 - adăugarea celulei B }
 new(R);
 R^.Info:='B';
 R^.Prec:=V; { crearea legăturii B -> A }
 R^.Urm:=nil;
 V^.Urm:=R; { crearea legăturii A -> B }
 V:=R; { actualizarea adresei vârfului }
 { 4 - adăugarea celulei C }
 new(R);
 R^.Info:='C';
 R^.Prec:=V; { crearea legăturii C -> B }
 R^.Urm:=nil;
 V^.Urm:=R; { crearea legăturii B -> C }
 V:=R; { actualizarea adresei vârfului }
 { 5 - adăugarea celulei D }
 new(R);
 R^.Info:='D';
 R^.Prec:=V; { crearea legăturii D -> C }
 R^.Urm:=nil;
 V^.Urm:=R; { crearea legăturii C -> D }
 V:=R; { actualizarea adresei vârfului }

 { afișarea listei create }
 R:=P;
 while R<>nil do begin
 writeln(R^.Info);
 R:=R^.Urm
 end;

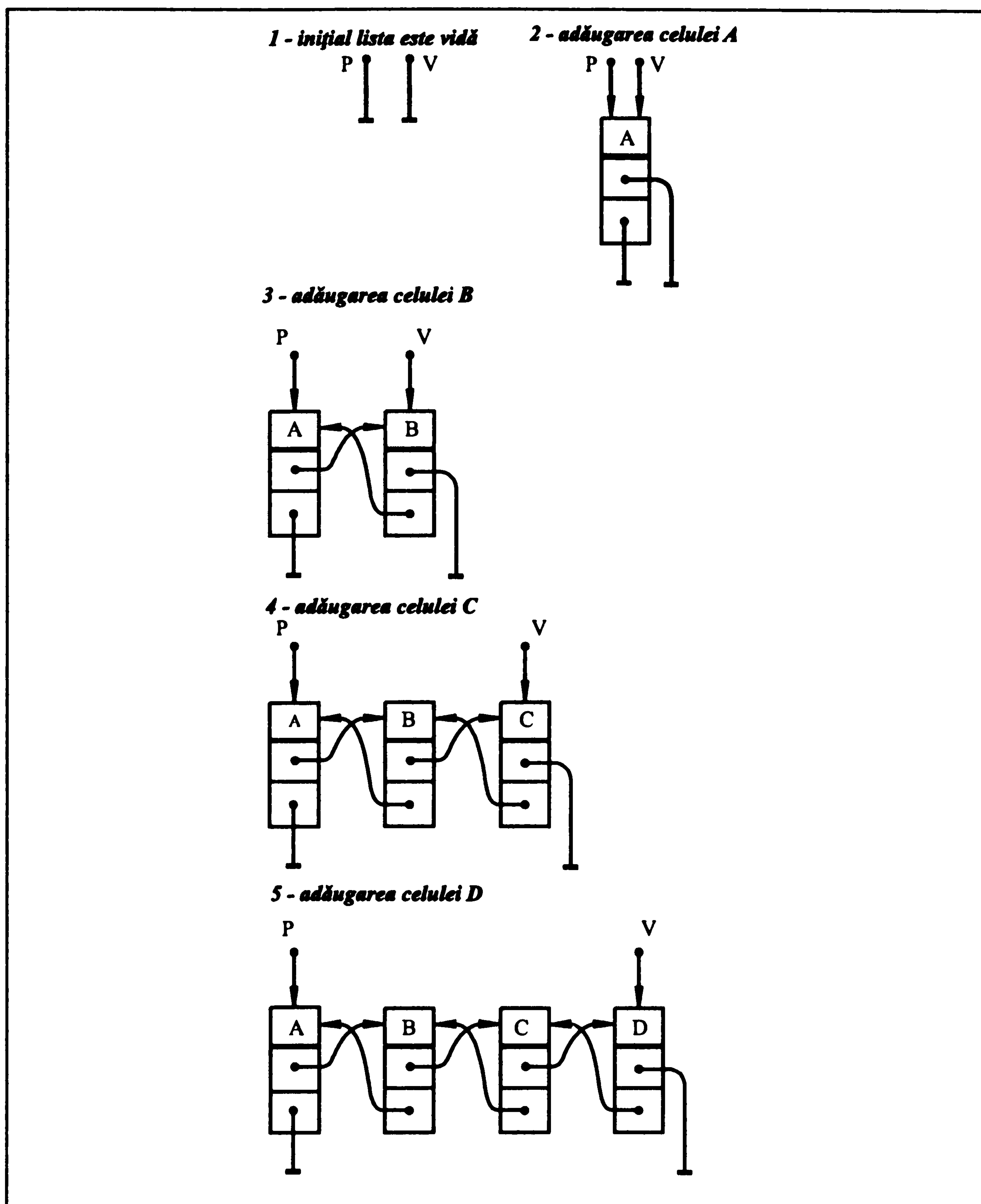
 readln;
end.

```

Procesul de construire a listei în studiu este prezentat în *fig. 6.7*. Asupra listelor bidirecționale se efectuează următoarele operații:

- parcurgerea listei de la bază la vîrf;
- parcurgerea listei de la vîrf la bază;
- căutarea unui element, identificat prin valoarea sa;
- includerea unui element în listă;

- excluderea unui element din listă;
- reordonarea elementelor listei conform unui anumit criteriu;
- concatenarea listelor;
- descompunerea unei liste în mai multe liste ș.a.



*Fig. 6.7. Crearea listei bidirecționale*

Ca și în cazul listelor unidirecționale, includerea și excluderea elementelor, concatenarea și descompunerea listelor constă în crearea și eventual distrugerea celulelor sau a legăturilor respective. Evident, parcurgerea listelor bidirecționale în ambele direcții se realizează mai

simplu datorită faptului că pentru fiecare celulă din listă se cunoaște nu numai adresa celulei următoare, dar și adresa celulei precedente.

*Exemplu:*

```
Program P126;
{ Crearea și parcurgerea unei liste bidirecționale }
type AdresaCelula = ^Celula;
 Celula = record
 Info : string;
 Urm, Prec : AdresaCelula;
 end;

var P, { adresa de bază }
 V : AdresaCelula; { adresa vârfului }

procedure Creare;
var R : AdresaCelula;
begin
 P := nil; V := nil;
 writeln('Dați lista:');
 while not eof do
 begin
 new(R);
 readln(R^.Info);
 R^.Prec := nil;
 R^.Urm := nil;
 if P = nil then begin P := R; V := R end
 else begin V^.Urm := R; R^.Prec := V;
V := R end;
 end;
 end; { Creare }

procedure Parcurgere;
var R : AdresaCelula;
begin
 writeln('Parcurgere de la bază la vîrf:');
 R := P;
 while R <> nil do
 begin
 writeln(R^.Info);
 R := R^.Urm;
 end;
 writeln('Parcurgere de la vîrf la bază:');
 R := V;
 while R <> nil do
 begin
 writeln(R^.Info);
 R := R^.Prec;
 end;
 readln;
end; { Parcurgere }
```



```

begin
 Creare;
 Parcurgere;
end.

```

## Întrebări și exerciții

- ❶ Cum se definesc datele necesare pentru crearea unei liste bidirecționale? Care este destinația câmpurilor *Prec* și *Urm* ale unei celule?
- ❷ Scrieți un program care creează lista bidirecțională din *fig. 6.6*, adăugînd cîte un element la baza listei.
- ❸ Completați programul P126 cu procedurile necesare pentru includerea și excluderea elementelor din lista bidirecțională creată.
- ❹ Elaborați o procedură care:

- a) transformă o listă unidirecțională într-o listă bidirecțională;
- b) transformă o listă bidirecțională într-o listă unidirecțională.

Sînt oare posibile aceste transformări fără a dubla volumul de memorie alocat listei inițiale?

- ❺ Scrieți o procedură care parcurge o listă unidirecțională de la vîrf la bază. Comparați complexitatea acestei proceduri cu complexitatea secvenței respective de instrucțiuni din programul P126.
- ❻ Care sînt avantajele și neajunsurile listelor bidirecționale comparativ cu listele unidirecționale? Comparativ cu tablourile unidimensionale?
- ❼ În jurul arbitrilor sînt aranjați în cerc  $N$  jucători. Începînd cu jucătorul  $m$ , cu rația  $k$ , jucătorii sînt eliminați. Elaborați un program care afișează pe ecran ordinea de ieșire din cerc.

*Indicație:* Jucătorii aranjați în cerc pot fi reprezentați printr-o listă bidirecțională în care sînt stabilite legături între vîrf și bază (listă inelară).

## 6.6. STIVA

Prin **stivă** (în limba engleză *stack*) înțelegem o listă unidirecțională cu proprietatea că operațiile de introducere și extragere a elementelor se fac la un singur capăt al ei. Poziția ocupată în stivă de ultimul element introdus poartă numele de **vîrf**. O stivă fără nici un element se numește **stivă vidă**.

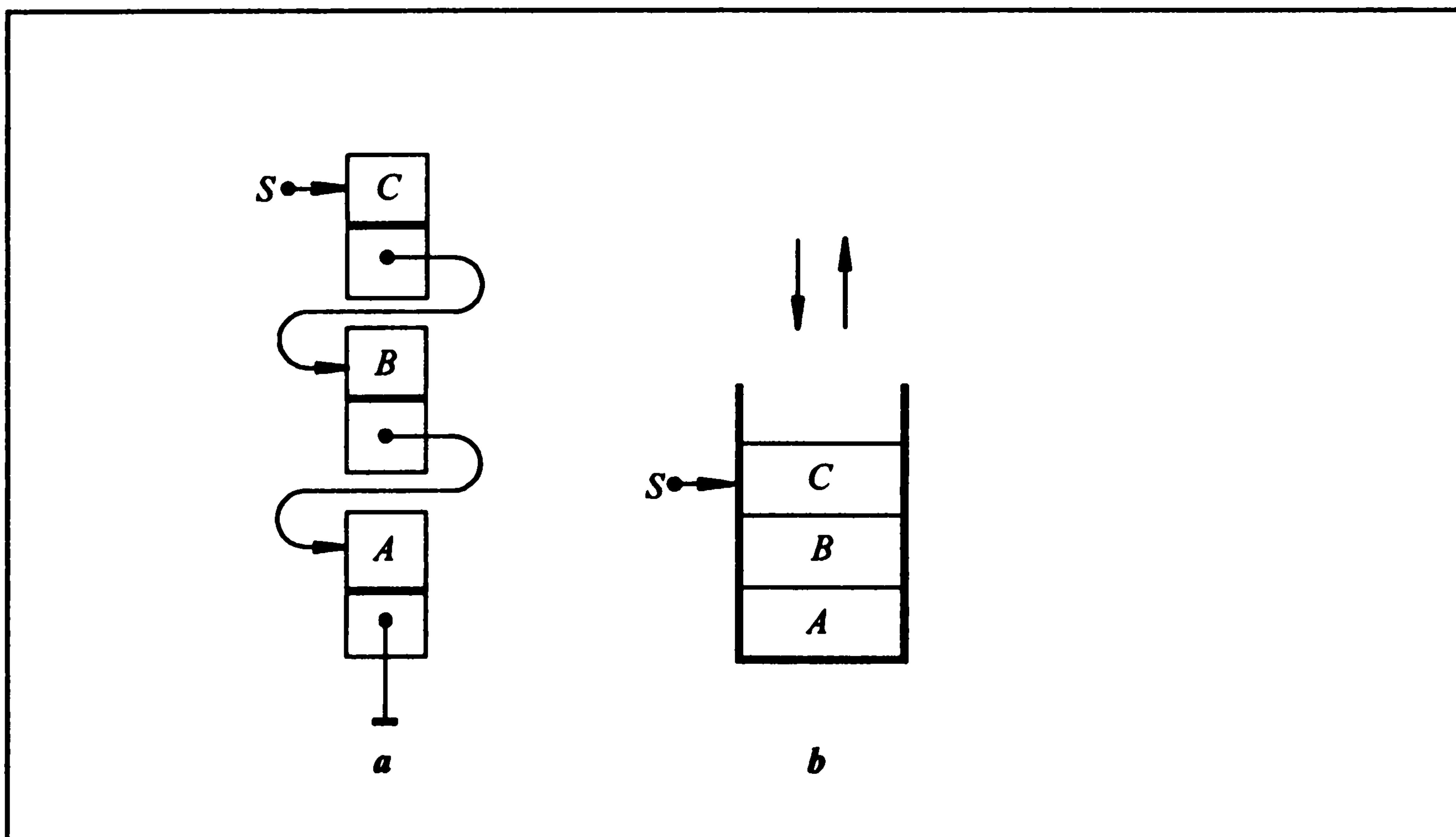
Pentru exemplificare, în *fig. 6.8* este prezentată o stivă care conține elementele A, B, C.

Datele necesare pentru crearea și prelucrarea unei stive pot fi definite prin declarații de forma:

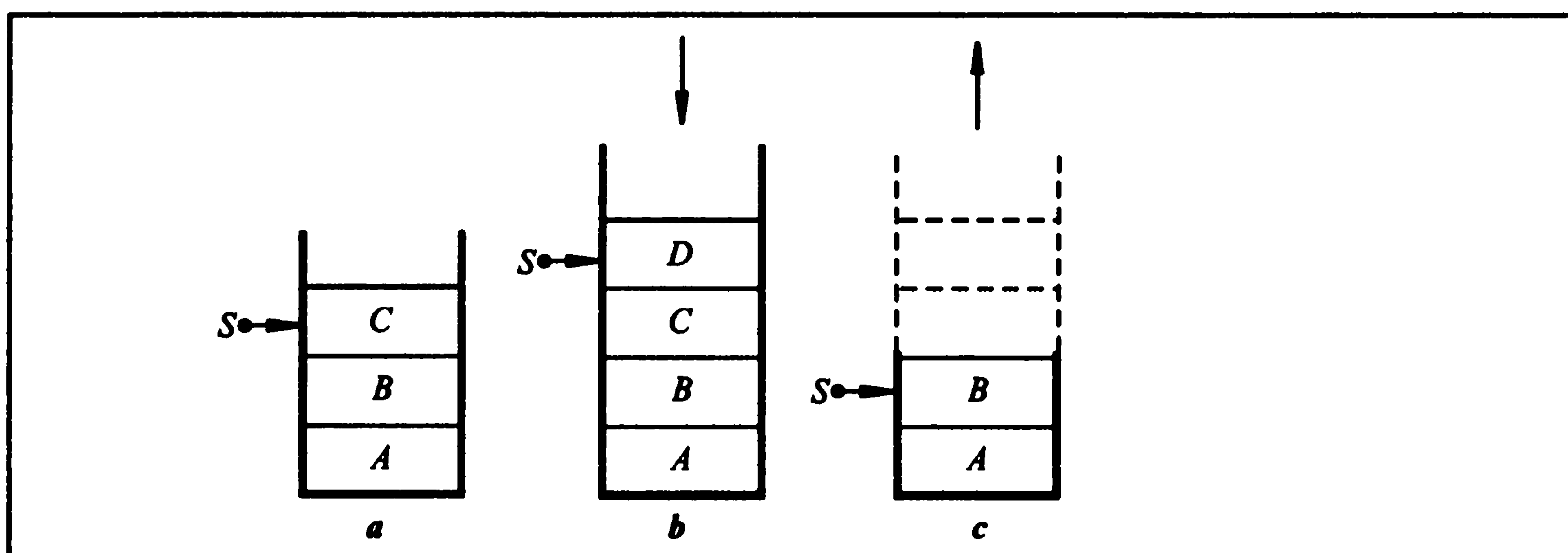
```

type AdresaCelula = ^Celula;
 Celula = record
 Info : string;
 Prec : AdresaCelula
 end;
var S : AdresaCelula;

```



*Fig. 6.8. Stiva:*  
*a* – reprezentarea detaliată; *b* – reprezentarea generalizată



*Fig. 6.9. Introducerea și extragerea elementelor din stivă:*  
*a* – stiva inițială; *b* – introducerea elementului D;  
*c* – extragerea elementelor D, C

Adresa vârfului stivei este memorată în variabila de tip *referință* *S*.  
 Adresa celulei precedente din stivă este memorată în câmpul *Prec*.

**Operația de introducere** a unui element în stivă (*fig. 6.9*) este efectuată de secvența de instrucțiuni:

```
new(R); { crearea unei celule }
{ încărcarea informației utile
 în câmpul R^.Info }
R^.Prec:=S; { crearea legăturii
 către celula precedentă din stivă }
S:=R; { actualizarea adresei vârfului }
```

unde *R* este o variabilă de tipul *AdresaCelula*.

**Extragerea unui element din stivă (fig. 6.9)** este efectuată de secvența:

```
R:=S; { memorarea adresei celulei extrase }
 { prelucrarea informației din câmpul R^.Info }
 S:=S^.Prec; { eliminarea celulei din stivă }
 dispose(R); { distrugerea celulei extrase }
```

*Exemplu:*

```
Program P127;
{ Crearea și prelucrarea unei stive }
type AdresaCelula=^Celula;
 Celula=record
 Info : string;
 Prec : AdresaCelula;
 end;
var S : AdresaCelula; { adresa vârfului }
 c : char;
procedure Introduc;
var R : AdresaCelula;
begin
 new(R);
 write('Dați elementul ce urmează');
 writeln(' să fie introdus:');
 readln(R^.Info);
 R^.Prec:=S;
 S:=R;
end; { Includ }

procedure Extrag;
var R : AdresaCelula;
begin
 if S=nil then writeln('Stiva este vidă')
 else begin
 R:=S;
 write('Este extras');
 writeln(' elementul:');
 writeln(R^.Info);
 S:=S^.Prec;
 dispose(R);
 end;
end; { Extrag }

procedure Afis;
var R : AdresaCelula;
begin
 if S=nil then writeln('Stiva este vidă')
 else begin
 writeln('Stiva include elementele:');
 R:=S;
```



```

 while R<>nil do begin
 writeln(R^.Info);
 R:=R^.Prec;
 end;

 end;
readln;
end; { Afis }
begin
 S:=nil; { inițial stiva este vidă }
 repeat
 writeln('Meniu:');
 writeln('I - Introducerea elementului');
 writeln('E - Extragerea elementului');
 writeln('A - Afișarea stivei pe ecran');
 writeln('O - Oprirea programului');
 write('Opțiunea='); readln(c);
 case c of
 'I' : Introduc;
 'E' : Extrag;
 'A' : Afis;
 'O' :
 else writeln('Opțiune necunoscută')
 end;
 until c='O';
end.

```

Stivele mai poartă și numele de liste *LIFO* (*last in, first out* — ultimul element care a intrat în stivă va fi primul care va ieși din ea) și sînt frecvent utilizate pentru alocarea dinamică a memoriei în cazul procedurilor și funcțiilor recursive. Evident, stivele pot fi simulate utilizînd tablourile unidimensionale **array**[ 1 .. *n*] **of** ..., însă o astfel de reprezentare este limitată de cele *n* componente ale tablourilor.

## Întrebări și exerciții

- ❶ Care este ordinea de introducere și extragere a datelor din stivă?
- ❷ De la tastatură se citesc mai multe numere naturale. Afișați numerele în studiu pe ecran în ordinea inversă citirii.
- ❸ În *fig. 6.10* este reprezentată schema de manevrare a vagoanelor de tren într-un depou.

Elaborați un program care citește de la tastatură și afișează pe ecran datele despre fiecare vagon intrat sau ieșit din depou. Datele în studiu includ:

- numărul de înmatriculare (*integer*);
- stația de înmatriculare (*string*);
- anul fabricării (1960 .. 2000);
- tipul vagonului (*string*);
- proprietarul vagonului (*string*);
- capacitatea de încărcare (*real*).

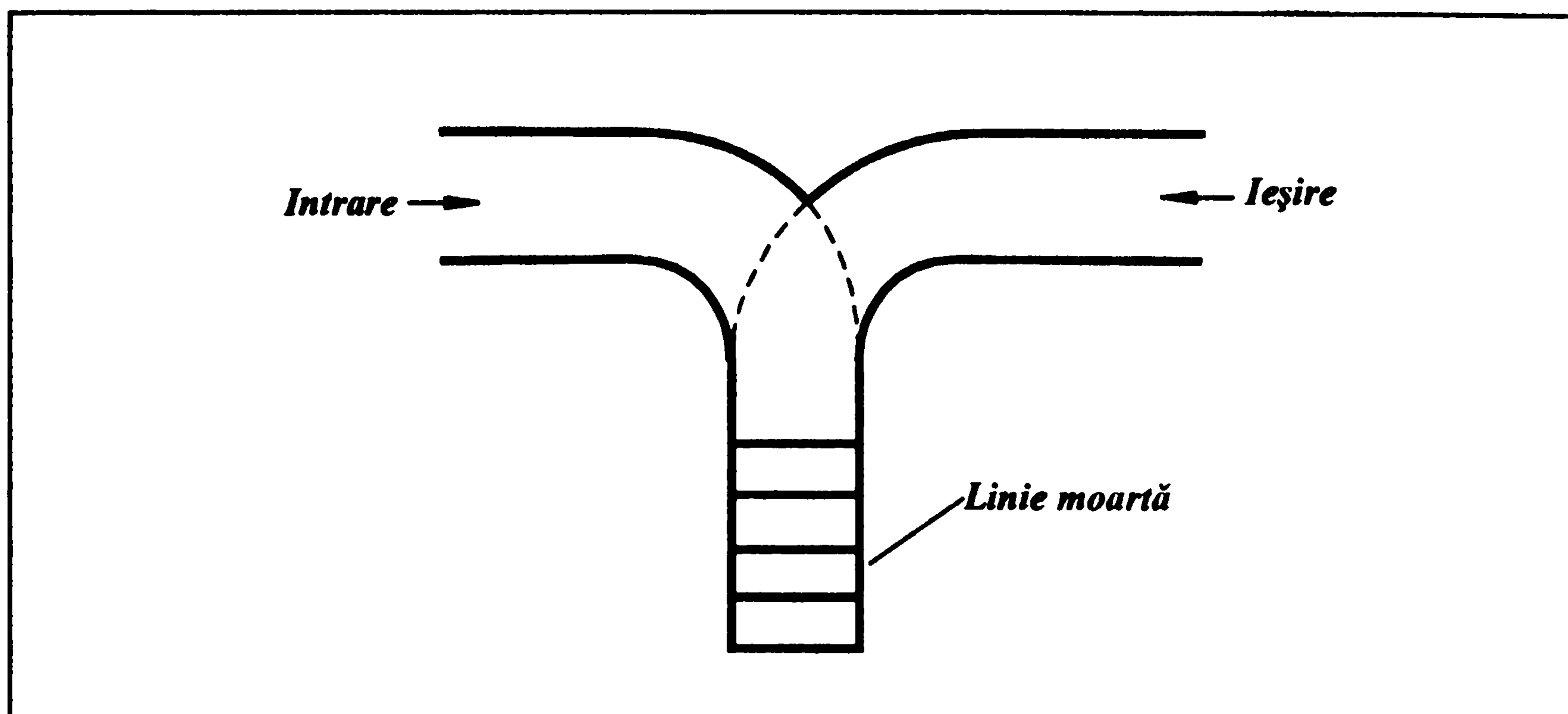


Fig. 6.10. Schema de manevrare a vagoanelor de tren

- ④ Pot fi oare folosite listele bidirecționale pentru crearea unei stive?
- ⑤ Colectivele temporare de muncă sînt formate și desființate în ordinea “ultimul angajat va fi primul care va fi concediat”. Elaborați un program care citește de la tastatură și afișează pe ecran datele despre fiecare persoană angajată sau concediată. Datele în studiu includ :
  - numele (string);
  - prenumele (string);
  - anul nașterii (1930..1985);
  - data angajării (ziua, luna, anul).
- ⑥ Se consideră șiruri finite de caractere formate din parantezele (, ), [, ], {, }. Un șir este **corect** numai atunci cînd el poate fi construit cu ajutorul următoarelor reguli:
  - a) șirul vid este corect;
  - b) dacă  $A$  este un șir corect, atunci  $(A)$ ,  $[A]$  și  $\{A\}$  sînt șiruri corecte;
  - c) dacă  $A$  și  $B$  sînt șiruri corecte, atunci  $AB$  este un șir corect.

De exemplu, șirurile ( ), [ ], { }, [( )], ((([ ])))([ ])) sînt corecte, iar șirurile ([, ( )][ ]{ }, ([ ]) nu sînt corecte. Elaborați un program care verifică dacă șirul citit de la tastatură este corect.

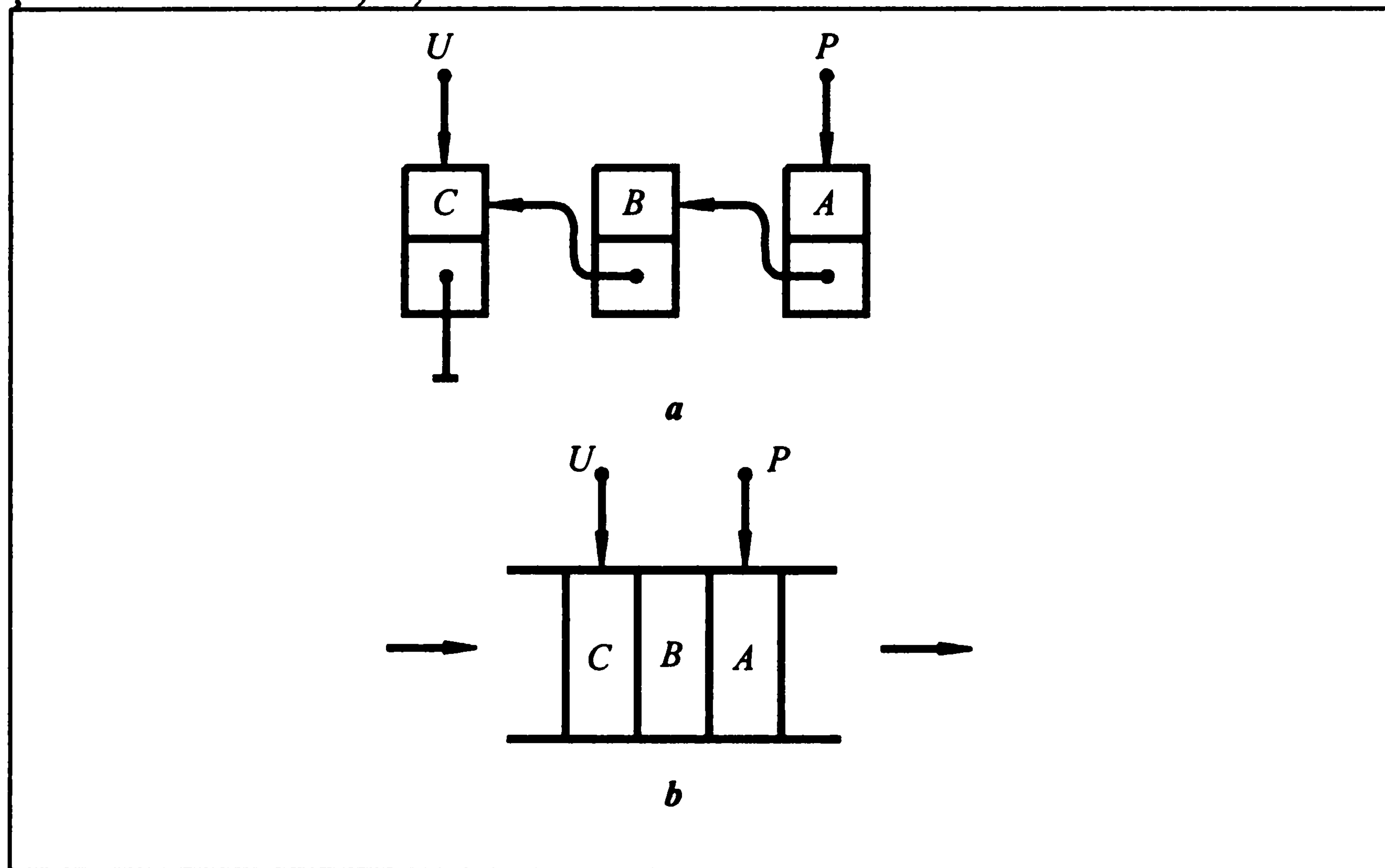
*Indicație.* Problema poate fi rezolvată printr-o singură parcurgere a șirului supus verificării. Dacă caracterul curent este (, [ sau {, el este depus în stivă. Dacă vârful stivei și caracterul curent formează una din perechile ( ), [ ] sau { }, paranteza respectivă este scoasă din stivă. În cazul unui șir corect, după examinarea ultimului caracter din șir stiva rămîne vidă.

- ⑦ Elementele stivei sînt memorate într-un tablou unidimensional. Elaborați procedurile necesare pentru introducerea și extragerea elementelor din stivă. Care sînt avantajele și neajunsurile acestei reprezentări? Se consideră că stiva conține cel mult 100 de elemente.

## 6.7. COZI

Prin **coadă** (în engleză *queue*) înțelegem o listă unidirecțională în care toate introducerile se efectuează la unul din capete, iar extragerile se efectuează la celălalt capăt. O coadă fără nici un element se numește **coadă vidă**.

Pentru exemplificare, în *fig. 6.11* este prezentată o coadă care conține elementele A, B, C.



*Fig. 6.11. Coada:*  
*a* – reprezentarea detaliată; *b* – reprezentarea generalizată

Datele necesare pentru crearea și prelucrarea unei cozi pot fi definite prin declarații de forma:

```
type AdresaCelula=^Celula;
 Celula=record
 Info : string;
 Urm : AdresaCelula
 end;
var P, U : AdresaCelula;
```

Adresa primului element din coadă este memorată în variabila de tip referință P, iar adresa ultimului element în variabila U. Adresa celulei următoare din coadă este memorată în câmpul Urm.

**Operația de introducere** a unui element (*fig. 6.12*) este efectuată de secvența de instrucțiuni:

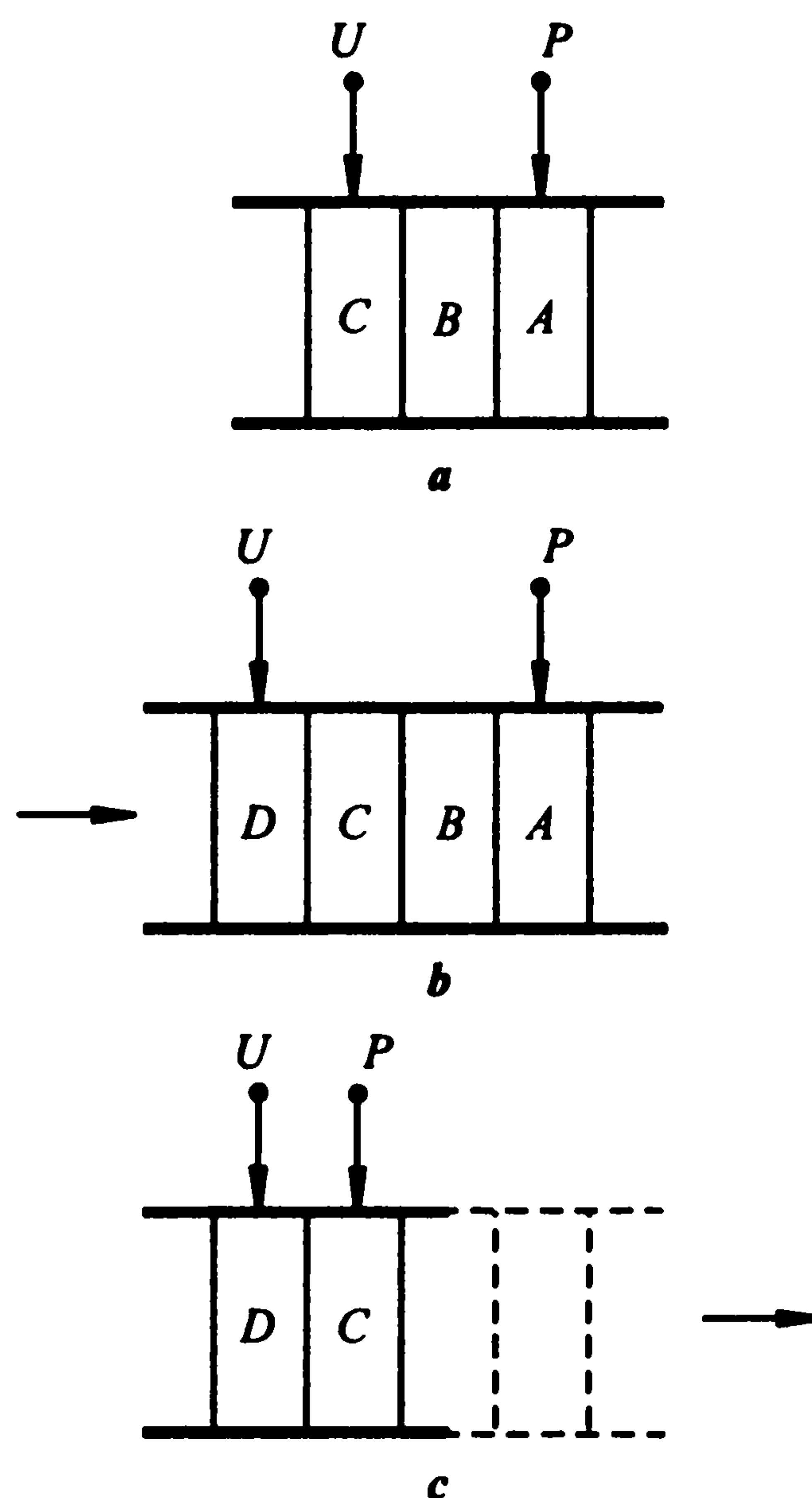
```
new (R); { crearea unei celule }
{ încărcarea informației utile
```



```

 în câmpul R^.Info }
R^.Urm:=nil; { înscrierea indicatorului
 "ultimul element" }
U^.Urm:=R; { adăugarea celulei la coadă }
U:=R; { actualizarea adresei ultimei celule }

```



*Fig. 6.12. Introducerea și extragerea elementelor din coadă:*  
*a – coada inițială; b – introducerea elementului D;*  
*c – extragerea elementelor A, B*

**Extragerea** unui element din coadă (*fig. 6.12*) este efectuată de secvența:

```

R:=P; { memorarea adresei primei celule }
{ prelucrarea informației din câmpul R^.Info }
P:=P^.Urm; { eliminarea primei celule }
dispose(R); { distrugerea celulei extrase }

```

*Exemplu:*

```

Program P128;
{ Crearea și prelucrarea unei cozi }
type AdresaCelula=^Celula;
 Celula=record
 Info : string;
 Urm : AdresaCelula;

```

```

 end;

var P, { adresa primului element }
 U : AdresaCelula; { adresa ultimului element }
 c : char;

procedure Introduc;
var R : AdresaCelula;
begin
 new(R);
 write('Dați elementul ce urmează');
 writeln(' să fie introdus:');
 readln(R^.Info);
 R^.Urm:=nil;
 if P=nil then begin P:=R; U:=R end
 else begin U^.Urm:=R; U:=R end;
end; { Introduc }

procedure Extrag;
var R : AdresaCelula;
begin
 if P=nil then writeln('Coadă este vidă')
 else begin
 R:=P;
 write('Este extras');
 writeln(' elementul:');
 writeln(R^.Info);
 P:=P^.Urm;
 dispose(R);
 end;
end; { Extrag }

procedure Afis;
var R : AdresaCelula;
begin
 if P=nil then writeln('Coadă este vidă')
 else begin
 write('Coadă include');
 writeln(' elementele:');
 R:=P;
 while R<>nil do
 begin
 writeln(R^.Info);
 R:=R^.Urm;
 end;
 end;
 end;

 readln;
end; { Afis }

begin
 P:=nil; U:=nil; { inițial coada este vidă }
 repeat

```

```

writeln('Meniu:');
writeln('I - Introducerea elementului;');
writeln('E - Extragerea elementului;');
writeln('A - Afişarea cozii la ecran;');
writeln('O - Oprirea programului');
write('Opţiunea='); readln(c);
case c of
 'I' : Introduc;
 'E' : Extrag;
 'A' : Afis;
 'O' :
 else writeln('Opţiune necunoscută')
end;
until c='O';
end.

```

Cozile mai poartă numele de **liste *FIFO*** (*first in, first out* — primul element intrat în coadă va fi primul ieşit din coadă). Menţionăm că simularea cozilor cu ajutorul tablourilor unidimensionale este inefficientă din cauza migrării elementelor cozii spre ultima componentă a tabloului.

## Întrebări şi exerciţii

- ❶ Pot fi oare folosite listele bidirecţionale pentru crearea unei *cozi*? Care ar fi avantajele unei astfel de structuri de date?
- ❷ Elaboraţi o funcţie care returnează numărul elementelor unei cozi.
- ❸ Avioanele care solicită aterizarea pe o anumită pistă a unui aeroport formează un fir de aşteptare. Elaboraţi un program care citeşte de la tastatură şi afişează pe ecran datele despre fiecare avion care solicită aterizarea şi avionul care aterizează. Datele în studiu includ:
  - numărul de înmatriculare (*integer*);
  - tipul avionului (*string*);
  - numărul rutei (*integer*).
- ❹ Prin **coadă cu priorităţi** vom înţelege o coadă în care elementul de introdus se inserează nu după ultimul element al cozii, ci înaintea tuturor elementelor cu o prioritate mai mică. Priorităţile elementelor se indică prin numere întregi. Elaboraţi un program care:
  - a) creează o coadă cu priorităţi;
  - b) introduce în coadă elementele specificate de utilizator;
  - c) extrage elementele din coadă;
  - d) afişează coada cu priorităţi pe ecran.
- ❺ Se consideră liste bidirecţionale care îmbină proprietăţile stivei şi cozii: introducerile şi extragerile se fac la ambele capete. Elaboraţi un program care efectuează următoarele operaţii:
  - a) creare;
  - b) adăugare la dreapta;



- c) adăugare la stînga;
- d) ștergere la dreapta listei;
- e) ștergere la stînga listei;
- f) parcurgerea listei de la stînga la dreapta;
- g) parcurgerea listei de la dreapta la stînga.

## 6.8. ARBORI BINARI

Prin **nod** se înțelege o variabilă dinamică de tipul **record** care conține un câmp destinat memorării informațiilor utile și doi indicatori de adresă.

**Arborele binar** se definește recursiv după cum urmează:

- a) un nod este un arbore binar;
- b) un nod ce conține legături către alți doi arbori binari este un arbore binar.

Prin convenție, **arborele vid** nu conține nici un nod. Pentru exemplificare, în *fig. 6.13* este prezentat un arbore binar nodurile căruia conțin informația utilă A, B, C, D, E, F, G, H, I, J. Datele necesare pentru crearea și prelucrarea unui arbore binar pot fi definite prin declarații de forma:

```

type AdresaNod = ^Nod;
 Nod = record
 Info : string;
 Stg, Dr : AdresaNod
 end;
var T : AdresaNod;
```

Pentru a sublinia faptul că arborii binari sînt **structuri recursive** de date, declarațiile în studiu pot fi transcrise în forma:

```

type Arbore = ^Nod;
 Nod = record
 Info : string;
 Stg, Dr : Arbore;
 end;
var T : Arbore;
```

Nodul spre care nu este îndreptată nici o legătură se numește **rădăcină**. Adresa rădăcinii se memorează în variabila de tip referință T. În cazul unui arbore vid  $T = \text{nil}$ .

Cei doi arbori conectați la rădăcină se numesc **subarborele stîng** și **subarborele drept**. Adresa subarborelui stîng se memorează în câmpul Stg, iar adresa subarborelui drept — în câmpul Dr.

**Nivelul** unui nod este, prin convenție, 0 pentru nodul-rădăcină și  $i + 1$ , pentru nodul conectat la un nod de nivelul  $i$ . În mod obișnuit, în

reprezentarea grafică a unui arbore binar nodurile se desenează pe niveluri: rădăcina se află pe nivelul 0, vîrfurile conectate la rădăcină — pe nivelul 1 ș.a.m.d. (fig. 6.13). Nodurile de pe nivelul  $i + 1$ , conectate la un nod de pe nivelul  $i$ , se numesc **descendenții** acestuia. În fig. 6.13 nodul B este descendentul stîng, iar nodul C este descenden-

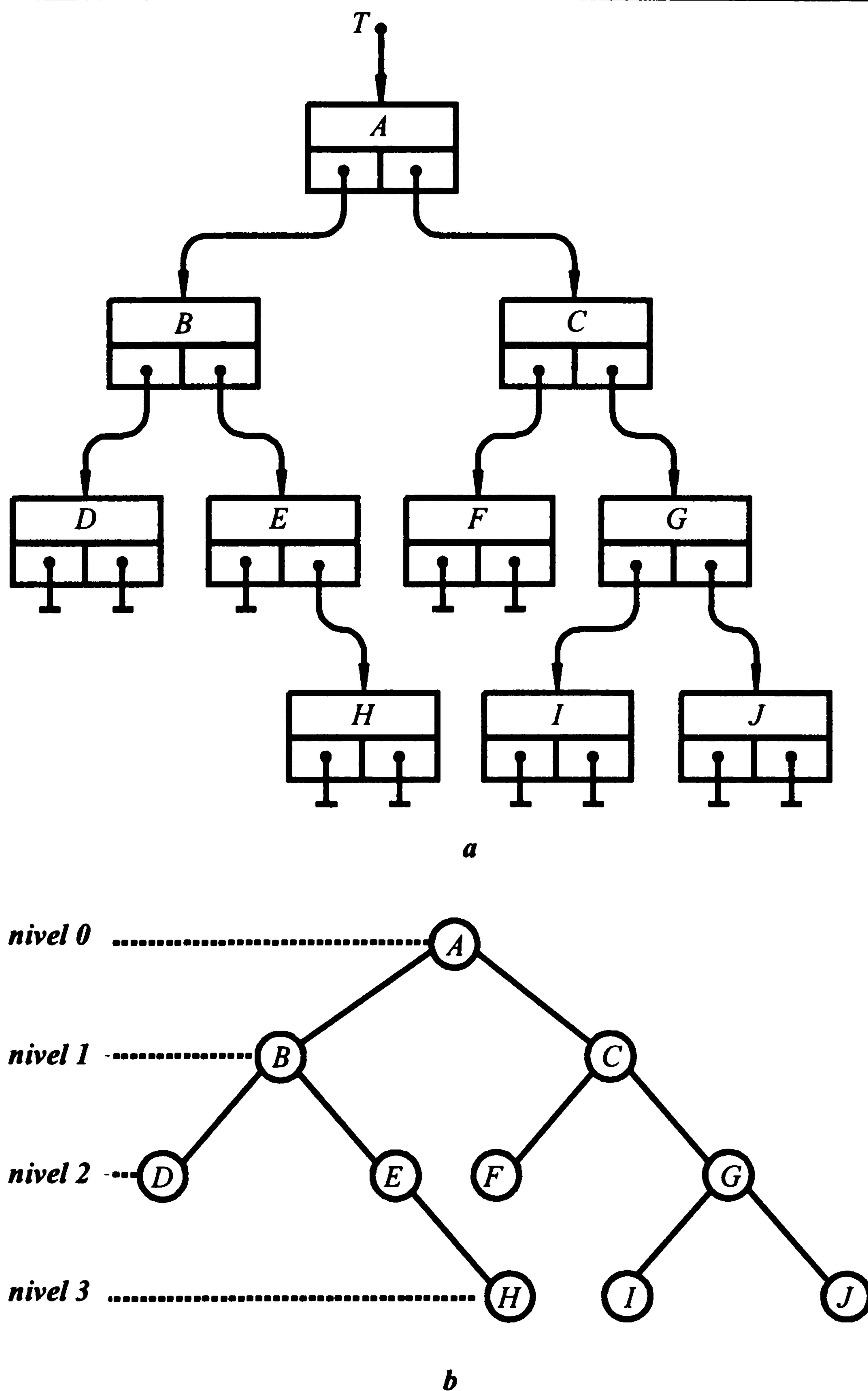


Fig. 6.13. Arbore binar:  
a – reprezentarea detaliată; b – reprezentarea generalizată



tul drept al nodului A; nodul D este descendentul stîng, iar nodul E — descendentul drept al nodului B ș.a.m.d.

Dacă un nod  $x$  este descendentul altui nod  $y$ , îl numim pe acesta din urmă **părintele** nodului  $x$ . În *fig. 6.13* nodul A este părintele nodurilor B și C; nodul B este părintele nodurilor D și E ș.a.m.d.

Un nod la care nu este conectat nici un subarbore este un **nod terminal**. În caz contrar nodul este **neterminal**. Prin **înălțimea** arborelui binar înțelegem numărul de nivel maxim asociat nodurilor terminale. Arborele din *fig. 6.13* are înălțimea 3; nodurile D, H, F, I și J sînt noduri terminale; nodurile A, B, C, E și G sînt noduri neterminale.

Arborii binari pot fi construiți în memoria calculatorului cu ajutorul algoritmilor iterativi sau algoritmilor recursivi.

**Algoritmul iterativ** creează nodurile în ordinea apariției lor pe niveluri:

- se creează nodul-rădăcină;
- nodul-rădăcină se introduce într-o coadă;
- pentru fiecare nod extras din coadă se creează, dacă există, descendentul stîng și descendentul drept;
- nodurile nou-create se introduc în coadă;
- procesul de construire a arborelui se încheie cînd coada devine vidă.

Nodurile arborelui din *fig. 6.13* vor fi create de algoritmul iterativ în ordinea: A, B, C, D, E, F, G, H, I, J.

Un algoritm similar poate fi utilizat pentru parcurgerea arborelui binar și afișarea nodurilor respective pe ecran:

- se creează o coadă care conține un singur element – nodul-rădăcină;
- fiecare nod extras din coadă este afișat pe ecran;
- descendenții nodului extras se introduc în coadă;
- procesul de afișare se încheie cînd coada devine vidă.

*Exemplu:*

```
Program P129;
{ Crearea unui arbore binar - iterație }
type AdresaNod = ^Nod;
 Nod = record
 Info : string;
 Stg, Dr : AdresaNod
 end;

 AdresaCelula = ^Celula;
 Celula = record
 Info : AdresaNod;
 Urm : AdresaCelula
 end;
```



```

var T : AdresaNod; { rădăcina }
 Prim, { primul element din coadă }
 Ultim : AdresaCelula; { ultimul element din coadă }

procedure IntroduInCoadă(Q : AdresaNod);
var R : AdresaCelula;
begin
 new(R);
 R^.Info:=Q;
 R^.Urm:=nil;
 if Prim=nil then begin Prim:=R; Ultim:=R end
 else begin Ultim^.Urm:=R; Ultim:=R end;
end; { IntroduInCoadă }

procedure ExtrageDinCoadă(var Q : AdresaNod);
var R : AdresaCelula;
begin
 if Prim=nil then writeln('Coadă este vidă')
 else begin
 R:=Prim;
 Q:=R^.Info;
 Prim:=Prim^.Urm;
 dispose(R);
 end;
end; { ExtrageDinCoadă }

procedure CreareArboreBinar;
var R, Q : AdresaNod;
 s : string;
begin
 T:=nil; { inițial arborele este vid }
 Prim:=nil; Ultim:=nil; { inițial coada este vidă }
 writeln('Dați rădăcina:'); readln(s);
 if s<>' ' then
 begin
 new(R); { crearea rădăcinii }
 R^.Info:=s;
 T:=R; { inițializarea adresei rădăcinii }
 IntroduInCoadă(T);
 end;
 while Prim<>nil do { cât coada nu e vidă }
 begin
 ExtrageDinCoadă(R);
 writeln('Dați descendenții nodului', R^.Info);
 write(' stîng: '); readln(s);
 if s=' ' then R^.Stg:=nil
 else
 begin
 new(Q); R^.Stg:=Q;
 Q^.Info:=s;

```

```

 IntroduInCoada(Q);
 end; { else }
write(' drept: '); readln(s);
if s='' then R^.Dr:=nil
 else
 begin
 new(Q); R^.Dr:=Q;
 Q^.Info:=s;
 IntroduInCoada(Q);
 end; { else }
 end; { while }
end; { CreareArboreBinar }
procedure AfisareArboreBinar;
var R : AdresaNod;
begin
 if T=nil then writeln('Arbore vid')
 else
 begin
 writeln('Arborele este format din:');
 Prim:=nil; Ultim:=nil;
 IntroduInCoada(T);
 while Prim<>nil do
 begin
 ExtrageDinCoada(R);
 writeln('Nodul ', R^.Info);
 write(' descendenti: ');
 if R^.Stg=nil then write('nil, ')
 else begin
 write(R^.Stg^.Info, ', ');
 IntroduInCoada(R^.Stg);
 end;

 if R^.Dr=nil then writeln('nil')
 else begin
 writeln(R^.Dr^.Info);
 IntroduInCoada(R^.Dr);
 end;
 end; { while }
 end; { else }
 readln;
 end; { AfisareArboreBinar }

begin
 CreareArboreBinar;
 AfisareArboreBinar;
end.

```

Informația utilă asociată fiecărui nod se citește de la tastatură. Absența descendentului se semnalează prin apăsarea tastei <ENTER> (programul citește de la tastatură un șir vid de caractere). Menționăm

că coada creată de programul P129 nu conține nodurile propriu-zise, ci adresele acestor noduri.

**Algoritmul recursiv** construiește arborii binari urmînd direct definiția respectivă:

- se creează nodul-rădăcină;
- se construiește subarborele stîng;
- se construiește subarborele drept.

Nodurile arborelui binar din *fig. 6.13* vor fi create de algoritmul recursiv în ordinea: A, B, D, E, H, C, F, G, I, J.

*Exemplu:*

```
Program P130;
{ Crearea unui arbore binar - recursie }
type Arbore = ^Nod;
 Nod = record
 Info : string;
 Stg, Dr : Arbore
 end;

var T : Arbore; { rădăcina }

function Arb : Arbore;
{ crearea arborelui binar }
var R : Arbore;
 s : string;
begin
 readln(s);
 if s = '' then Arb := nil
 else begin
 new(R);
 R^.Info := s;
 write('Dați descendentul stîng');
 writeln(' al nodului ', s, ':');
 R^.Stg := Arb;
 write('Dați descendentul drept');
 writeln(' al nodului ', s, ':');
 R^.Dr := Arb;
 Arb := R;
 end;
end; { Arb }

procedure AfisArb(T : Arbore; nivel : integer);
{ afisarea arborelui binar }
var i : integer;
begin
 if T <> nil then
 begin
 AfisArb(T^.Stg, nivel+1);
```



```

 for i:=1 to nivel do write(' ');
 writeln(T^.Info);
 AfisArb(T^.Dr, nivel+1);
 end;
end; { AfisareArb }

begin
 writeln('Dați rădăcina:');
 T:=Arb;
 AfisArb(T, 0);
 readln;
end.

```

Funcția Arb citește de la tastatură informația utilă asociată nodului în curs de creare. Dacă se citește un șir vid, nu se creează nici un nod și funcția returnează valoarea **nil**. În caz contrar, funcția creează un nod, înscrie șirul de caractere în câmpul Info și returnează adresa nodului. În momentul când trebuie completate câmpurile Stg (adresa subarborelui stîng) și Dr (adresa subarborelui drept) funcția se autoapelează, trecînd astfel la construcția subarborelui respectiv.

Procedura AfisArb afișează arborele binar pe ecran. Se afișează subarboarele stîng, rădăcina și apoi subarboarele drept. Nivelul fiecărui nod este redat prin inserarea numărului respectiv de spații.

Comparînd programele P129 și P130, se observă că prelucrarea structurilor recursive de date, și anume, a arborilor binari, este mai naturală și mai eficientă în cazul utilizării unor algoritmi recursivi.

Arborii binari au numeroase aplicații, una dintre cele specifice fiind reprezentarea expresiilor în scopul prelucrării acestora în traducătoarele limbajelor de programare.

## Întrebări și exerciții

- ❶ Cum se definește un *arbore binar*? Explicați termenii: *rădăcină*, *subarboarele stîng*, *subarboarele drept*, *descendent*, *nivel*, *nod terminal*, *nod neterminal*, *înălțimea arborelui binar*.
- ❷ Formulați algoritmi iterativi destinați creării și afișării arborilor binari.
- ❸ Cum se construiește un arbore binar cu ajutorul algoritmului recursiv?
- ❹ Elaborați un program care construiește arborele genealogic propriu pe parcursul a trei sau patru generații. Nodul-rădăcină conține numele, prenumele și anul nașterii, iar nodurile descendente conțin datele respective despre părinți.
- ❺ Cum trebuie modificată procedura AfisArb din programul P130 pentru ca arborele binar să fie afișat în ordinea: subarboarele drept, nodul-rădăcină, subarboarele stîng?
- ❻ Scrieți o funcție recursivă care returnează numărul nodurilor unui arbore binar. Transcrieți această funcție într-o formă nerecursivă.
- ❼ Organizarea unui turneu “prin eliminare” este redată cu ajutorul unui arbore binar. Nodurile arborelui în studiu conțin următoarea informație:



- numele (**string**);
- prenumele (**string**);
- data nașterii (ziua, luna, anul);
- cetățenia (**string**).

Fiecărui jucător îi corespunde un nod terminal, iar fiecărui meci – un nod ne-terminal. În fiecare nod ne-terminal se înscriu datele despre câștigătorul meciului la care au participat cei doi jucători din nodurile descendente. Evident, rădăcina arborelui va conține datele despre câștigătorul turneului.

Scrieți un program care creează în memoria calculatorului și afișează pe ecran arborele unui turneu prin eliminare.

*Indicație:* Se pornește de la o listă a jucătorilor. Câștigătorii meciurilor din prima etapă se includ într-o altă listă. În continuare se formează lista câștigătorilor meciurilor din etapa a doua ș.a.m.d.

- ⑧ Cum trebuie modificată funcția `Arb` din programul P130 pentru ca arborele binar să se construiască în ordinea: A, C, G, J, I, F, B, E, H, D?
- ⑨ Funcția `Arb` din programul P130 construiește arborii binari în ordinea: nodul-rădăcină, subarborele stîng, subarborele drept. Scrieți o procedură nerecursivă care construiește arborii binari în aceeași ordine.

*Indicație:* Se utilizează o stivă elementele căreia sînt noduri. Inițial stiva va conține numai nodul-rădăcină. Pentru fiecare nod din vârful stivei se va construi subarborele stîng, iar apoi — subarborele drept. Nodurile nou-create se introduc în stivă. După construirea subarborelui drept nodul respectiv este scos din stivă.

## 6.9. PARCURGEREA ARBORILOR BINARI

**Operațiile** care se pot efectua asupra arborilor binari se împart în două mari categorii:

- operații care modifică structura arborelui (inserarea sau eliminarea unui nod);
- operații care păstrează intactă structura arborelui (căutarea unei informații, tipărirea informațiilor asociate unui nod etc.). Una din problemele care apar în mod frecvent la efectuarea acestor operații este necesitatea de a parcurge sau a traversa arborele binar.

Prin **parcurerea unui arbore** se înțelege examinarea în mod sistematic a nodurilor sale astfel încît informația din fiecare nod să fie prelucrată o singură dată. Există trei modalități de parcuregere a arborilor binari: nodurile pot fi vizitate în preordine, inordine și postordine. Aceste trei metode sînt definite recursiv: dacă arborele este vid, atunci el este parcurs fără a se face nimic; astfel parcuregere se face în trei etape.

**Parcuregere în preordine** sau traversarea *RSD*:

- 1) se vizează rădăcina;
- 2) se traversează subarborele stîng;
- 3) se traversează subarborele drept.

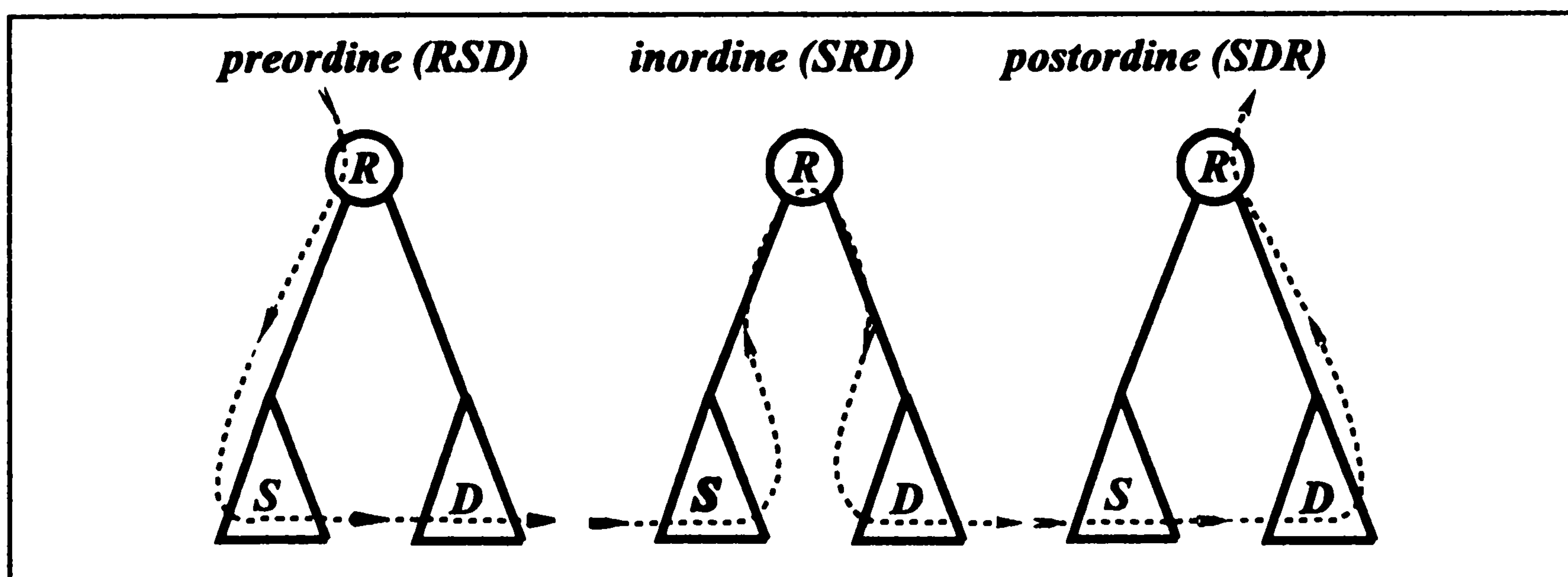


Fig. 6.14. Metode de parcurgere a arborilor binari

**Parcuregerea în inordine** sau traversarea *SRD*:

- 1) se traversează subarborele stîng;
- 2) se vizitează rădăcina;
- 3) se traversează subarborele drept.

**Parcuregerea în postordine** sau traversarea *SDR*:

- 1) se traversează subarborele stîng;
- 2) se traversează subarborele drept;
- 3) se vizitează rădăcina.

Notățiile *RSD*, *SRD* și *SDR* reprezintă ordinea în care vor fi vizate rădăcina (*R*), subarborele stîng (*S*) și subarborele drept (*D*). Metodele de parcurgere a arborilor binari sînt ilustrate în *fig. 6.14*.

Pentru arborele din *fig. 6.13* parcurgerea în preordine furnizează nodurile în ordinea:

A, B, D, E, H, C, F, G, I, J;

parcuregerea în inordine furnizează nodurile în ordinea:

D, B, E, H, A, F, C, I, G, J;

iar parcurgerea în postordine conduce la:

D, H, E, B, F, I, J, G, C, A.

Prezentăm mai jos un program PASCAL de parcurgere a unui arbore binar după toate cele trei metode.

```

Program P131;
{ Parcuregerea arborelui binar }
type Arbore=^Nod;
 Nod=record
 Info : string;
 Stg, Dr : Arbore
 end;
var T : Arbore; { rădăcina }

```



```

function Arb : Arbore;
 { crearea arborelui binar }
var R : Arbore;
 s : string;
begin
 readln(s);
 if s='' then Arb:=nil
 else begin
 new(R);
 R^.Info:=s;
 write('Dați descendentul stîng');
 writeln(' al nodului ', s, ':');
 R^.Stg:=Arb;
 write('Dați descendentul drept');
 writeln(' al nodului ', s, ':');
 R^.Dr:=Arb;
 Arb:=R;
 end;
end; { Arb }

procedure AfisArb(T : Arbore; nivel : integer);
 { afișarea arborelui binar }
var i : integer;
begin
 if T<>nil then
 begin
 AfisArb(T^.Stg, nivel+1);
 for i:=1 to nivel do write(' ');
 writeln(T^.Info);
 AfisArb(T^.Dr, nivel+1);
 end;
end; { AfisareArb }

procedure Preordine(T : Arbore);
 { traversare RSD }
begin
 if T<>nil then begin
 writeln(T^.Info);
 Preordine(T^.Stg);
 Preordine(T^.Dr)
 end;
end; { Preordine }

procedure Inordine(T : Arbore);
 { traversare SRD }
begin
 if T<>nil then begin
 Inordine(T^.Stg);
 writeln(T^.Info);
 Inordine(T^.Dr)
 end;
end;

```

```

 end;
end; { Preordine }

procedure Postordine(T : Arbore);
{ traversare SDR }
begin
 if T<>nil then begin
 Postordine(T^.Stg);
 Postordine(T^.Dr);
 writeln(T^.Info)
 end;
end; { Postordine }

begin
 writeln('Dați rădăcina:');
 T:=Arb;
 AfisArb(T, 0);
 readln;
 writeln('Parcurgere în preordine:');
 Preordine(T);
 readln;
 writeln('Parcurgere în inordine:');
 Inordine(T);
 readln;
 writeln('Parcurgere în postordine:');
 Postordine(T);
 readln;
end.

```

Menționăm că funcția Arb creează nodurile, parcurgând arborele binar în curs de construcție în preordine. Procedura AfisArb afișează nodurile parcurgând arborele binar în inordine.

## Întrebări și exerciții

- ① Ce operații pot fi efectuate asupra arborilor binari?
- ② Explicați metodele de parcurgere a arborilor binari. Dați exemple.
- ③ Scrieți listele de noduri obținute în urma celor trei metode de parcurgere a arborelui binar din *fig. 6.15*.
- ④ Transcrieți procedurile Preordine, Inordine și Postordine din programul P131 în formă nerecursivă.
- ⑤ Scrieți un subprogram care returnează înălțimea arborelui binar.
- ⑥ Elaborați un program care afișează pe ecran toate nodurile aflate pe un nivel dat într-un arbore binar.
- ⑦ Elaborați o procedură recursivă care parcurge un arbore binar în ordinea:
  - a) *RDS* (rădăcina — subarborele drept — subarborele stîng);
  - b) *DRS* (subarborele drept — rădăcina — subarborele stîng);
  - c) *DSR* (subarborele drept — subarborele stîng — rădăcina).

Transcrieți procedura elaborată într-o formă nerecursivă.

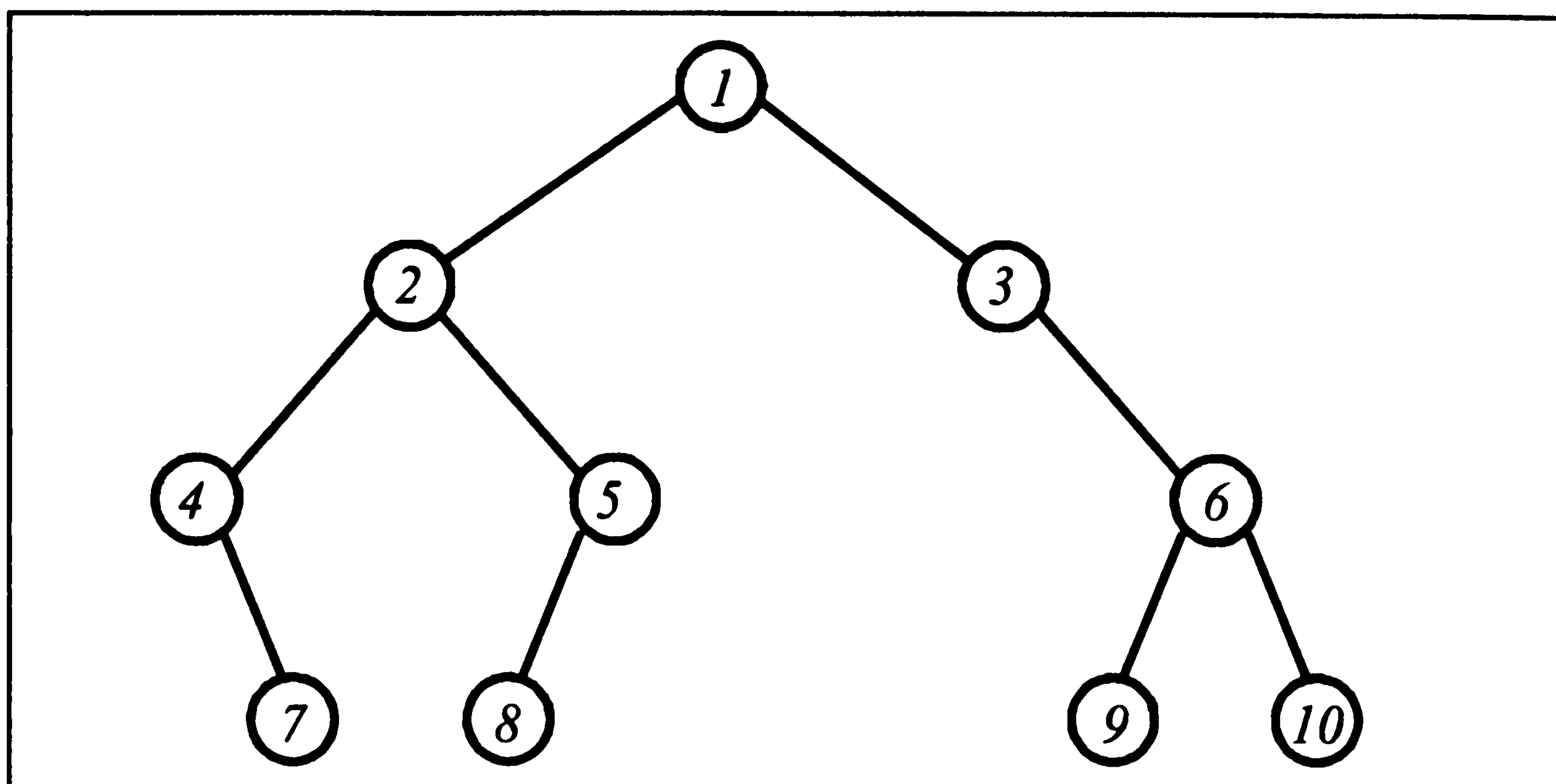


Fig. 6.15. Arbore binar

- ⑧ Scrieți un subprogram care afișează la ecran nivelul fiecărui nod dintr-un arbore binar.
- ⑨ Se dă un arbore binar în care nodurile terminale reprezintă numere întregi, iar cele neterminale operațiile binare  $+$ ,  $-$ ,  $*$ ,  $\text{mod}$ ,  $\text{div}$ . Arborele în studiu poate fi considerat ca o reprezentare a unei expresii aritmetice. Valoarea acestei expresii se calculează efectuând operația din nodul-rădăcină asupra valorilor subexpresiilor reprezentate de subarborele stîng și subarborele drept. Scrieți o funcție care returnează valoarea expresiilor aritmetice reprezentate prin arbori binari.
- ⑩ Se consideră expresiile aritmetice formate din operanzi și operatorii binari  $+$ ,  $-$ ,  $*$ ,  $/$ . Operanzii sînt variabile numele cărora este format dintr-o singură literă și constante alcătuite dintr-o cifră. Fiecărei expresii aritmetice  $i$  se poate asocia un arbore binar după cum urmează:
  - a) expresiei aritmetice formate dintr-un singur operand  $i$  se asociază un arbore binar format doar din nodul ce conține operandul respectiv;
  - b) expresiei aritmetice de forma  $E_1 \circ E_2$ , unde  $E_1$  și  $E_2$  sînt expresii aritmetice,  $i$  se asociază un arbore binar care are în nodul-rădăcină operatorul  $\circ$ , ca subarbore stîng arborele asociat expresiei  $E_1$ , iar ca subarbore drept arborele asociat expresiei  $E_2$ .

Valoarea expresiei se calculează efectuînd operația din nodul-rădăcină asupra valorilor subexpresiilor reprezentate de subarborele stîng și subarborele drept. Scrieți un program care:

- a) construiește arbori binari asociați expresiilor aritmetice citite de la tastatură;
- b) evaluează expresiile aritmetice reprezentate prin arborii binari.

*Indicație.* Algoritmul va urma definiția recursivă a arborelui în studiu. Ca operator curent " $\circ$ " se poate desemna orice operator  $+$ ,  $-$  din expresia supusă prelucrării. Operatorii  $*$ ,  $/$  pot fi desemnați ca operatori curenți numai cînd expresia supusă prelucrării nu conține operatorii  $+$ ,  $-$ .



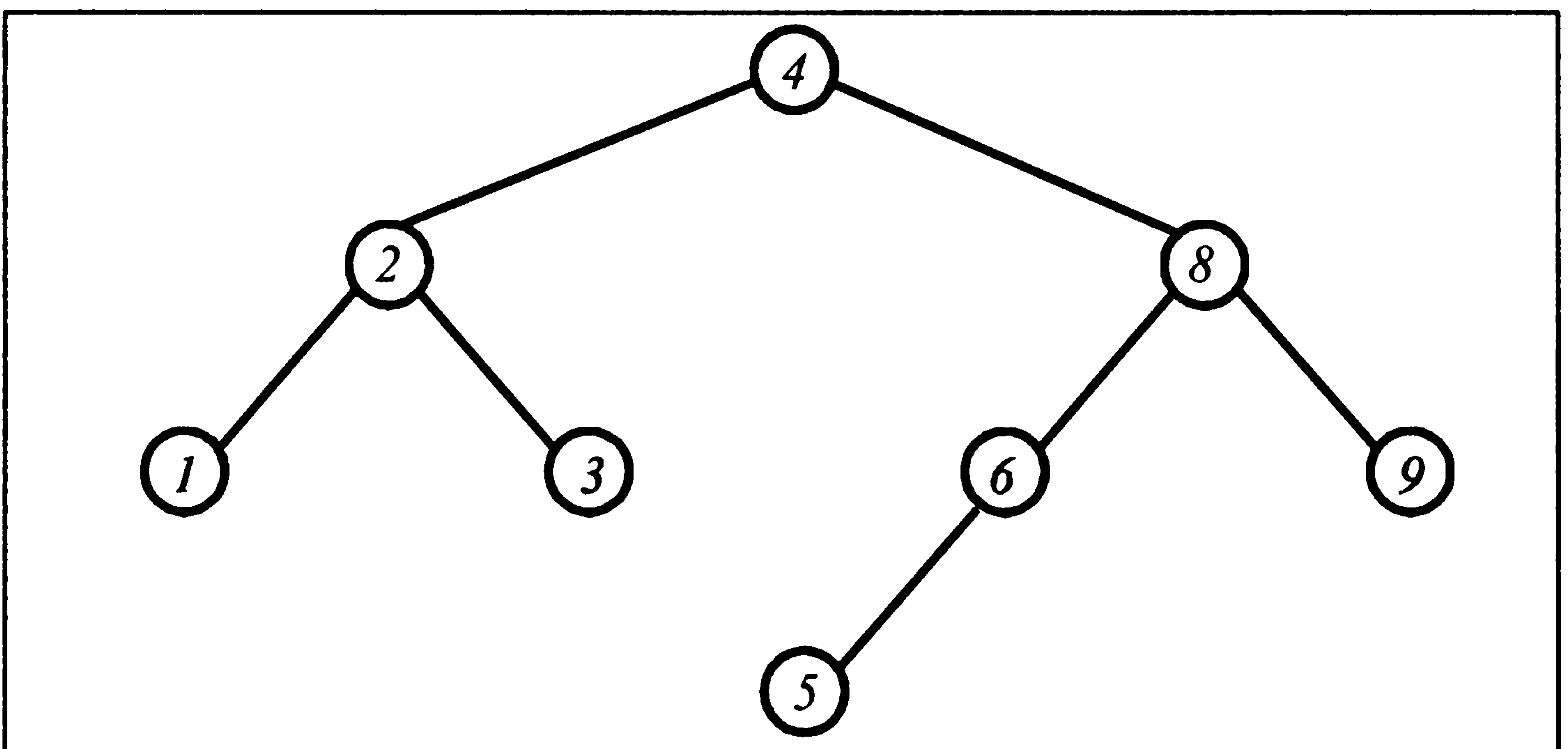
## 6.10. ARBORI BINARI DE CĂUTARE

Arborii binari sînt utilizați frecvent pentru reprezentarea datelor structurate, componentele cărora pot fi regăsite prin intermediul unei chei. În acest caz fiecare nod conține un câmp *Cheie* care identifică informația asociată nodului. Cheia unui nod este unică în arbore și, pentru simplificare, este reprezentată printr-un întreg.

Prin **arbore binar de căutare** se înțelege un arbore binar în care cheia oricărui nod este mai mare decît toate cheile nodurilor din subarboarele stîng și mai mică decît toate cheile nodurilor din subarboarele drept.

Din definiție rezultă că parcurgerea în inordine (subarboarele stîng — rădăcina — subarboarele drept) a unui arbore binar de căutare va furniza nodurile în ordine crescătoare a cheilor, motiv pentru care astfel de arbori se mai numesc **arbori binari de căutare-sortare**.

Un arbore binar de căutare este prezentat în *fig. 6.16*. Pentru simplificare sînt indicate numai cheile asociate nodurilor respective. Parcurgerea în inordine furnizează nodurile în ordinea: 1, 2, 3, 4, 5, 6, 8, 9.



*Fig. 6.16.* Arbore binar de căutare

Datele necesare pentru crearea și prelucrarea unui arbore binar de căutare pot fi definite prin declarații de forma:

```
type Arbore = ^Nod;
 Nod = Record
 Cheie : Integer;
 Info : String;
 Stg, Dr : Arbore
 end;
var T : Arbore;
```

**Operațiile** executate frecvent asupra unui arbore binar de căutare sînt căutarea și inserarea unui nod.

**Căutarea** nodului ce conține o anumită cheie se realizează foarte simplu cu ajutorul recursiei:

- dacă nodul-rădăcină conține cheia necesară, căutarea se termină cu succes;

- dacă nu, în funcție de rezultatul comparării cheilor respective, căutarea continuă în subarborele stîng sau în subarborele drept.

Dacă subarborele respectiv este vid, atunci căutarea eșuează.

De exemplu, pentru arborele din *fig. 6.16* și cheia 5, ordinea de vizitare a nodurilor va fi: 4, 8, 6, 5.

**Inserarea** unui nod ce conține o anumită cheie se realizează în mod similar:

- dacă arborele este vid, el se substituie cu nodul care urmează să fie inserat;

- în caz contrar, în funcție de rezultatul comparării cheilor respective, inserarea se va face în subarborele stîng sau subarborele drept.

De exemplu, la inserarea unui nod cu cheia 7 în arborele din *fig. 6.16* ordinea de vizitare a nodurilor va fi: 4, 8, 6. Nodul de inserat va deveni subarborele drept al nodului 6.

Programul care urmează implementează principalele operații realizate cu arborii binari de căutare.

```
Program P132;
{ Arbori binari de căutare }
type Arbore = ^Nod;
 Nod = record
 Cheie : integer;
 Info : string;
 Stg, Dr : Arbore
 end;
var T, { rădăcina }
 P : Arbore;
 ch : integer;
 inf : string;
 c : Char;

procedure Cautare(T : Arbore; ch : integer;
 var P : Arbore);
begin
 if T=nil then P:=nil
 else if ch=T^.Cheie then P:=T
 else if ch<T^.Cheie then
 Cautare(T^.Stg, ch, P)
 else
 Cautare(T^.Dr, ch, P);
```

```

end; { Cautare }

procedure Inserare(var T:Arbore; ch:integer;
 inf : string);
var Q : Arbore;
begin
 if T=nil then begin
 new(Q);
 Q^.Cheie:=ch;
 Q^.Info:=inf;
 Q^.Stg:=nil;
 Q^.Dr:=nil;
 T:=Q;
 end
 else if ch=T^.Cheie then
 writeln('Nodul există deja')
 else if ch<T^.Cheie
 then Inserare(T^.Stg, ch, inf)
 else Inserare(T^.Dr, ch, inf);
end; { Inserare }

procedure AfisArb(T:Arbore; nivel : integer);
 { afişarea arborelui binar }
var i : integer;
begin
 if T<>nil then
 begin
 AfisArb(T^.Stg, nivel+1);
 for i:=1 to nivel do write(' ');
 writeln(T^.Cheie, '_', T^.Info);
 AfisArb(T^.Dr, nivel+1);
 end;
end; { AfisareArb }

procedure Inordine(T : Arbore);
 { traversare SRD }
begin
 if T<>nil then begin
 Inordine(T^.Stg);
 writeln(T^.Cheie, '_', T^.Info);
 Inordine(T^.Dr)
 end;

end; { Inordine }

begin
 T:=nil; { inițial arborele este vid }
 repeat
 writeln('Meniu:');
 writeln('C - Căutare;');
 writeln('I - Inserare;');
 writeln('A - Afișare');

```



```

writeln('P - Parcurgere în inordine;');
writeln('O - Oprirea programului.');
```

**case** c **of**

    'C' : **begin**

        write('Cheia='); readln(ch);

        Cautare(T, ch, P);

**if** P=**nil** **then**

            writeln('Nod inexistent')

**else**

            writeln(P^.Info);

**end**;

    'I' : **begin**

        write('Cheia='); readln(ch);

        write('Info='); readln(inf);

        Inserare(T, ch, inf);

**end**;

    'A' : AfisArb(T, 0);

    'P' : Inordine(T);

    'O' :

**else** writeln('Opțiune necunoscută');

**end**; { **case** }

**until** c='O';

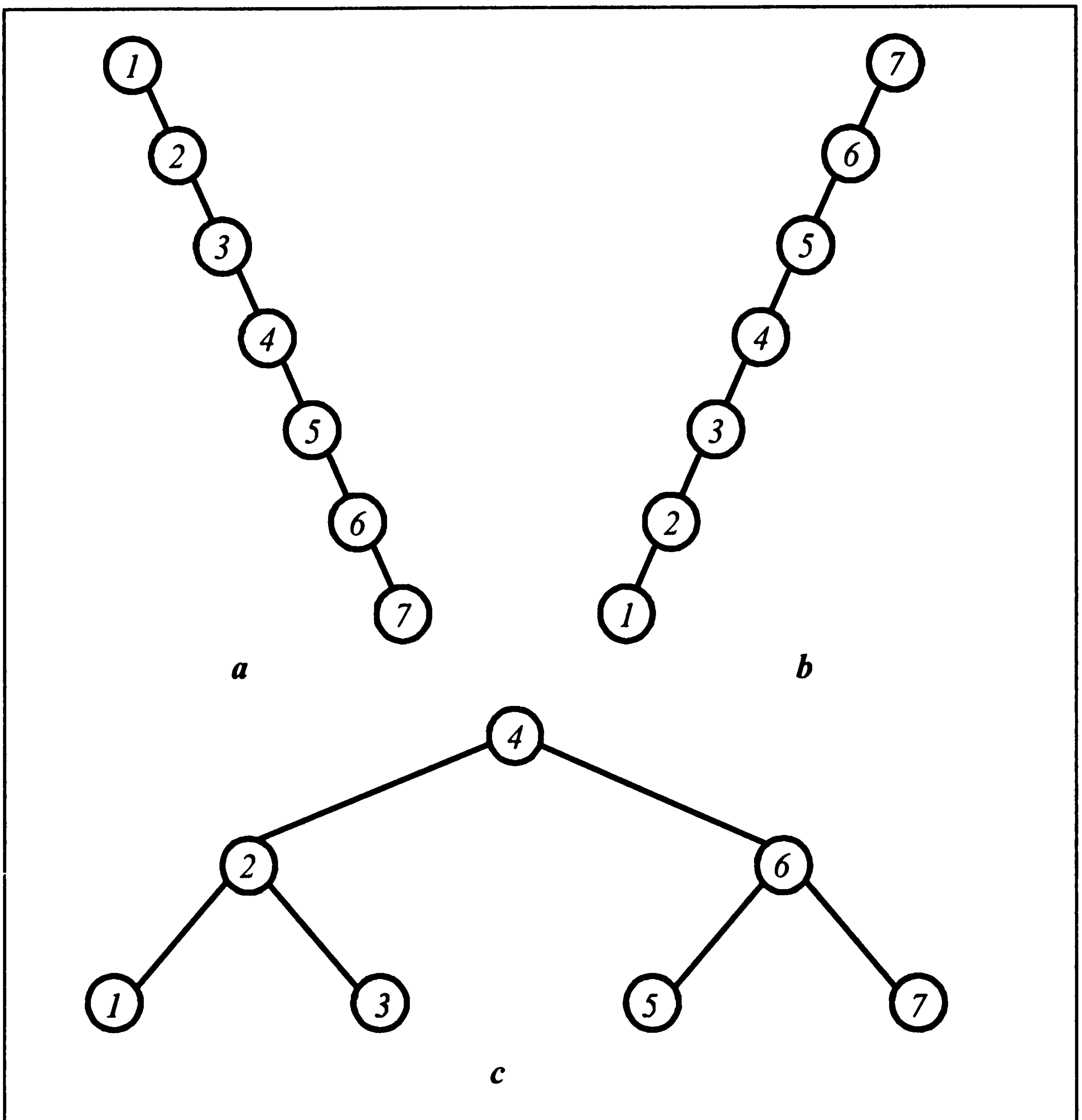
**end**.

Procedura Cautare returnează prin parametrul-variabilă P adresa nodului ce conține cheia ch specificată de utilizator.

Procedura Inserare include în arbore un nod și înscrie în câmpurile acestuia cheia ch și informația utilă inf. Subliniem faptul că adresa T a arborelui de căutare se transmite în procedura Inserare nu prin parametru-valoare, ci prin parametru-variabilă. Utilizarea parametrului-variabilă este vitală, întrucât substituirea subarborelui vid se realizează prin înlocuirea valorii **nil** din unul din câmpurile de adresă Stg, Dr ale nodului de pe nivelul precedent cu adresa nodului de inserat.

Avantajul arborilor binari de căutare constă în numărul redus de operații de căutare necesare pentru găsirea informației utile asociate unui nod. Evident, în procesul de căutare numărul nodurilor vizitate nu depășește valoarea  $h + 1$ , unde  $h$  este înălțimea arborelui. Menționăm că în cazul listelor numărul de căutări poate să atingă numărul  $n$  de celule din listă. Obişnuit,  $h \ll n$ .

Înălțimea arborilor binari de căutare construiți prin inserări consecutive de noduri depinde de ordinea inserării. În cazul cel mai rău, când nodurile se inserează în ordine crescătoare (descrescătoare), arborele degenerază, în fond, într-o listă unidirecțională (fig. 6.17). În majoritatea aplicațiilor se consideră că apariția unor astfel de cazuri este improbabilă.



*Fig. 6.17.* Arbori binari de căutare  
construiți prin inserări consecutive în ordine:  
*a* – crescătoare; *b* – descrescătoare; *c* – definită prin metoda înjumătățirii

În general, degenerarea arborelui de căutare într-o listă unidirecțională poate fi evitată, inserînd nodurile într-o anumită ordine definită prin **metoda înjumătățirii**. Conform acestei metode, șirul format din cheile nodurilor de inserat se ordonează crescător (descrescător). Primul se inserează nodul din mijlocul șirului. În continuare se inserează nodurile, cheile cărora se află în mijlocul subșirului stîng și subșirului drept ș.a.m.d.

De exemplu, în cazul cheilor 1, 2, 3, 4, 5, 6, 7 nodurile pot fi inserate în ordinea 4, 2, 6, 1, 3, 5, 7 (*fig. 6.17*). Se observă că înălțimea arborilor din *fig. 6.17a* și *6.17b* este  $h = 6$ , iar înălțimea arborelui din *fig. 6.17c* este  $h = 2$ .



## Întrebări și exerciții

- ❶ Este oare arborele binar din *fig. 6.15* un arbore binar de căutare?
- ❷ Ce operații pot fi executate asupra unui arbore binar de căutare? Cum se execută aceste operații?
- ❸ Transcrieți procedurile Inserare și Cautare din programul P132 într-o formă nerecursivă.
- ❹ Într-un registru sînt incluse următoarele date despre fiecare elev: numele și prenumele, data nașterii (ziua, luna, anul), nota medie. Scrieți un program care construiește și afișează pe ecran arborele binar de căutare ce folosește ca cheie:

- a) numele și prenumele;
- b) data nașterii;
- c) nota medie.

Pentru simplificare se consideră că nu există înregistrări cu chei identice.

- ❺ **Arborele lexicografic** este un arbore binar de căutare în care cheia reprezintă un cuvînt (o secvență nevidă formată din literele alfabetului latin), iar informația asociată nodului — numărul de apariții al cuvîntului respectiv într-un text. Scrieți un program care construiește arborele lexicografic al unui fișier text. Se consideră că literele mari și mici sînt identice. Afișați pe ecran arborele lexicografic al programului elaborat.
  - ❻ Arborele binar de căutare este construit prin inserarea consecutivă a  $n$  noduri. Ordinea de inserare este definită prin metoda înjumătățirii. Evaluați înălțimea arborelui și numărul de operații necesare pentru găsirea unui anumit nod din arbore.
  - ❼ **Arborele echilibrat** este un arbore binar de căutare organizat astfel încît, pentru orice nod din arbore  $|h(Stg) - h(Dr)| \leq 1$ , unde  $h(Stg)$  este înălțimea subarborelui stîng, iar  $h(Dr)$  este înălțimea subarborelui drept. Scrieți un program care:
- a) verifică dacă arborele binar de căutare este echilibrat;
  - b) transformă, dacă e necesar, orice arbore binar de căutare într-un arbore echilibrat.
- ❽ Pentru a micșora timpul de căutare, informația despre fișierele discului magnetic se păstrează în memoria calculatorului în forma unui arbore binar de căutare. Fiecărui fișier îi corespunde un nod care conține următoarele date:

- numele fișierului;
- data creării (ziua, luna, anul);
- ora creării (ore, minute, secunde);
- lungimea fișierului.

Scrieți un program care construiește și afișează pe ecran arborele binar de căutare ce folosește ca cheie:

- a) numele fișierului;
- b) data și ora creării.

- ❾ Scrieți un subprogram care elimină din arborele binar de căutare nodul ce conține o anumită cheie. După eliminare arborele rămas trebuie să fie arbore de căutare.

*Indicație.* Se disting următoarele situații:



- a) nodul de eliminat este un nod terminal — caz în care legătura către acest nod devine **nil**;
- b) nodul de eliminat subordonează un singur subarbore — caz în care legătura către nodul în studiu se redirecționează către subarborul respectiv;
- c) nodul de eliminat subordonează doi subarbori — caz în care se va încerca înlocuirea nodului cu cel mai din dreapta nod al subarborului stîng sau cu cel mai din stînga nod al subarborului drept.

⑩ Catalogul unei biblioteci este format din fișe. Fiecare fișă include următoarele date:

- numele și prenumele autorului;
- denumirea cărții;
- anul apariției;
- numărul de inventariere.

Fișele catalogului sînt ordonate alfabetic după numele și, dacă e necesar, prenumele autorului. Elaborați un program care construiește un arbore binar de căutare și afișează pe ecran:

- a) toate cărțile scrise de un anumit autor;
- b) numărul de inventariere al cărții specificate prin numele autorului, denumirea cărții și anul apariției.

Programul va asigura includerea și excluderea anumitor fișe din catalog.

## 6.11. ARBORI DE ORDINUL $m$

Se consideră variabile dinamice de tipul **record** care au în câmpul legăturilor  $m \geq 2$  indicatori de adresă. Ca și în cazul arborilor binari vom numi astfel de variabile **noduri**.

Arborele de ordinul  $m$  se definește recursiv după cum urmează:

- a) un nod este un arbore de ordinul  $m$ ;
- b) un nod ce conține cel mult  $m$  legături către alți arbori de ordinul  $m$  este un arbore de ordinul  $m$ .

Se consideră că în arbore există cel puțin un nod care subordonează exact  $m$  subarbori. Prin convenție, **arborele vid** nu conține nici un nod.

Arborii de ordinul 2 se numesc **arbori binari** și au fost studiați în paragrafele precedente. Arborii de ordinul 3, 4, 5 ș.a.m.d. se numesc **arbori multicăi** (în engleză *multiway tree*).

Pentru exemplificare, în *fig. 6.18* este prezentat un arbore de ordinul 4. Evident, pentru arborii multicăi termenii rădăcină, subarbore, nivel, părinte, descendent, nod terminal, nod neterminal, înălțime au aceeași semnificație ca și pentru arborii binari. Terminologia utilizată în structurile de date în studiu include chiar cuvinte ca *fiu, tată, frați, unchi, veri, străbunic* ș.a. cu înțeles similar celui din vorbirea curentă pentru noduri aflate pe diverse niveluri. Într-un limbaj simplist,

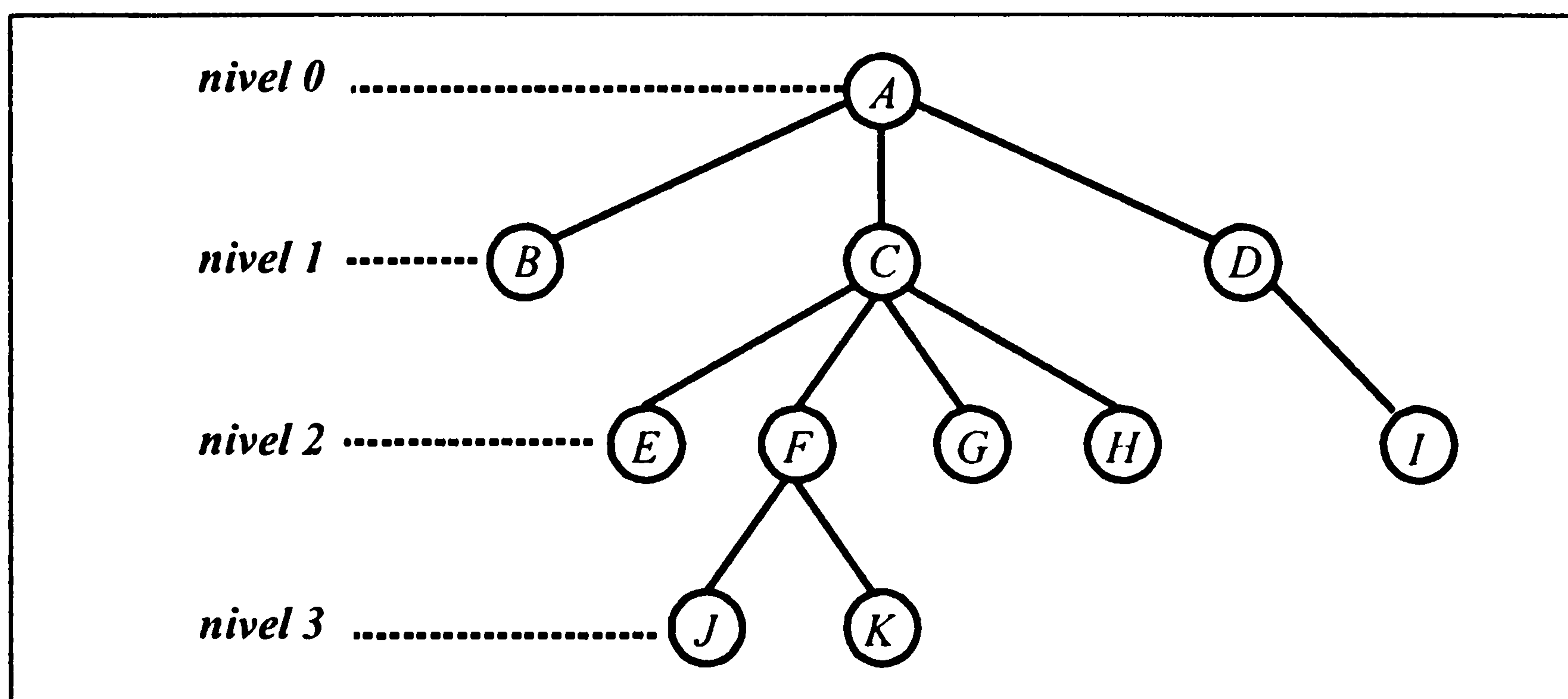


Fig. 6.18. Arbore de ordinul 4

structurile de date în studiu exprimă relații de “ramificare” între noduri, asemănătoare configurației arborilor din natură, cu deosebirea că în informatică arborii “cresc” în jos.

Datele necesare pentru crearea și prelucrarea unui arbore binar de ordinul  $m$  pot fi definite prin declarații de forma:

```

type Arbore = ^Nod;
 Nod = record;
 Info : string;
 Dsc : array [1..m] of Arbore
 end;
var T : Arbore;

```

Adresele descendenților unui nod se memorează în componentele  $Dsc[1]$ ,  $Dsc[2]$ , ...,  $Dsc[m]$  ale tabloului  $Dsc$ . Adresa rădăcinii se reține în variabila de tip referință  $T$ .

Cele mai uzuale metode de parcurgere a arborilor de ordinul  $m$  sînt parcurgerea în lățime și parcurgerea în adîncime.

**Parcurgerea în lățime** presupune vizitarea nodurilor în ordinea apariției lor pe niveluri. De exemplu, pentru arborele din *fig. 6.18* nodurile vor fi vizitate în ordinea: A, B, C, D, E, F, G, H, I, J, K.

În mod obișnuit, parcurgerea în lățime se realizează cu ajutorul unui algoritm iterativ care utilizează o structură auxiliară de date, și anume, o coadă formată din adresele nodurilor care vor fi vizitate.

**Parcurgerea în adîncime** se definește recursiv: dacă arborele este vid, el este parcurs fără a se face nimic; altfel se vizitează întîi rădăcina, apoi subarborii de la stînga la dreapta. Pentru arborele din *fig. 6.18* parcurgerea în adîncime furnizează nodurile în ordinea: A, B, C, E, F, J, K, G, H, D, I. Parcurgerea în adîncime se realizează foarte simplu cu ajutorul unui algoritm recursiv.



*Exemplu:*

```
Program P133;
{ Arbori de ordinul m }
const m=4;
type Arbore=^Nod;
 Nod=record
 Info : string;
 Dsc : array [1..m] of Arbore
 end;

 AdresaCelula=^Celula;
 Celula=record
 Info : Arbore;
 Urm : AdresaCelula
 end;

var T : Arbore; { rădăcina }
 Prim, { primul element din coadă }
 Ultim : AdresaCelula; { ultimul element
 din coadă }

procedure IntroduInCoadă(Q : Arbore);
var R : AdresaCelula;
begin
 new(R);
 R^.Info:=Q;
 R^.Urm:=nil;
 if Prim=nil then begin Prim:=R; Ultim:=R end
 else begin Ultim^.Urm:=R; Ultim:=R end;
end; { IntroduInCoadă }

procedure ExtrageDinCoadă(var Q : Arbore);
var R : AdresaCelula;
begin
 if Prim=nil then writeln('Coadă este vidă')
 else begin
 R:=Prim;
 Q:=R^.Info;
 Prim:=Prim^.Urm;
 dispose(R);
 end;
end; { ExtrageDinCoadă }

procedure CreareArbore(var T : Arbore);
var R, Q : Arbore;
 s : string;
 i : integer;
begin
 T:=nil; { inițial arborele este vid }
```



```

Prim:=nil; Ultim:=nil; { inițial coada este vidă }
writeln('Dați rădăcina: '); readln(s);
if s<>' ' then
begin
 new(R); { crearea rădăcinii }
 R^.Info:=s;
 T:=R; { inițializarea adresei rădăcinii }
 IntroduInCoada(T);
end;
while Prim<>nil do { cît coada nu e vidă }
begin
 ExtrageDinCoada(R);
 for i:=1 to m do R^.Dsc[i]:=nil;
 writeln('Dați descendenții nodului',R^.Info);
 i:=1; readln(s);
 while (i<=m) and (s<>' ') do
 begin
 new(Q); R^.Dsc[i]:=Q; Q^.Info:=s;
 IntroduInCoada(Q);
 i:=i+1; readln(s);
 end;
end;
end; { CreareArbore }

procedure AfisareArbore(T : Arbore);
var R : Arbore;
 i : integer;
begin
 if T=nil then writeln('Arbore vid')
 else
 begin
 writeln('Arborele este format din:');
 Prim:=nil; Ultim:=nil;
 IntroduInCoada(T);
 while Prim<>nil do
 begin
 ExtrageDinCoada(R);
 writeln('Nodul ', R^.Info);
 write(' Descendenți: ');
 for i:=1 to m do
 if R^.Dsc[i]<>nil then
 begin
 write(R^.Dsc[i]^Info, ' ');
 IntroduInCoada(R^.Dsc[i]);
 end; { then }
 writeln;
 end; { while }
 end; { else }
 readln;

```

```

end; { AfisareArbore }

procedure InLatime(T : Arbore);
var R : Arbore;
 i : integer;
begin
 if T<>nil then
 begin
 Prim:=nil; Ultim:=nil;
 IntroduInCoadă(T);
 while Prim<>nil do
 begin
 ExtrageDinCoadă(R);
 writeln(R^.Info);
 for i:=1 to m do
 if R^.Dsc[i]<>nil then
 IntroduInCoadă(R^.Dsc[i]);
 end; { while }
 end; { then }
 end; { InLatime }
 end; { InLatime }

procedure InAdincime(T : Arbore);
var i : integer;
begin
 if T<>nil then
 begin
 writeln(T^.Info);
 for i:=1 to m do InAdincime(T^.Dsc[i]);
 end;
 end; { InAdincime }

begin
 CreareArbore(T);
 AfisareArbore(T);
 writeln('Parcurgerea arborelui în lățime:');
 InLatime(T);
 readln;
 writeln('Parcurgerea arborelui în adâncime:');
 InAdincime(T);
 readln;
end.

```

Informația utilă asociată fiecărui nod se citește de la tastatură. Absența descendenților se semnalează prin apăsarea tastei <ENTER>. Menționăm că procedura CreareArbore creează nodurile parcurgând în lățime arborele în curs de construcție. Evident, procedura AfisareArbore vizitează nodurile în ordinea creării.

**Operațiile** frecvent efectuate asupra arborilor multicăi sînt: inserarea sau eliminarea unui nod, căutarea unei informații, prelucrarea in-



formațiilor utile asociate nodurilor ș.a. De obicei, arborii multicăi sînt utilizați în cazul aplicațiilor care implică prelucrarea unor mari cantități de date organizate ierarhic pe suporturile externe de informație. De exemplu, amintim modul de organizare a discurilor magnetice și optice în sistemele de operare MS-DOS, UNIX etc. Arborii în studiu sînt de asemenea utilizați în aplicațiile grafice pentru reprezentarea relațiilor dinamice dintre componentele imaginilor procesate.

## Întrebări și exerciții

- ❶ Dați exemple de arbori de ordinul 3, 5, 6.
- ❷ Cum se definește un arbore de ordinul  $m$ ? Ce operații pot fi efectuate asupra arborilor în studiu?
- ❸ Explicați metodele de parcurgere a arborilor multicăi. Dați exemple.
- ❹ Scrieți un program recursiv care construiește în memoria calculatorului un arbore multicăi. Informația utilă asociată nodurilor se citește de la tastatură.
- ❺ Scrieți o funcție care returnează:
  - a) numărul nodurilor unui arbore multicăi;
  - b) nivelul unui anumit nod din arbore;
  - c) înălțimea arborelui.
- ❻ Transcrieți procedura `InAdincime` din programul P133 într-o formă nerecursivă.
- ❼ Cum trebuie modificată procedura `InLatime` din programul P133 ca nodurile arborelui din *fig. 6.18* să fie vizitate în ordinea: A, D, C, B, I, H, G, F, E, K, J?
- ❽ Cum trebuie modificată procedura `InAdincime` din programul P133 pentru ca nodurile arborelui din *fig. 6.18* să fie vizitate în ordinea: A, D, I, C, H, G, F, K, J, E, B?
- ❾ Se dă un arbore multicăi, informațiile din noduri fiind șiruri de caractere. Să se afișeze pe ecran toate șirurile de caractere de lungime pară.
- ❿ Organizarea datelor de pe discurile magnetice este redată cu ajutorul unui arbore multicăi. Nodurile terminale reprezintă fișierele, iar nodurile neterminale — directoarele. Informația utilă asociată fiecărui nod include:
  - numele fișierului sau directorului (`string[ 8 ]`);
  - extensia (`string[ 3 ]`);
  - data și ora ultimei actualizări (respectiv ziua, luna, anul și ore, minute, secunde);
  - lungimea (`integer`);
  - `attribute('A', 'H', 'R', 'S')`.

Elaborați un program care simulează operațiile de căutare, creare și ștergere a fișierelor și directoarelor.

- ⓫ În unele cazuri ordinul  $m$  al arborelui multicăi nu este cunoscut în momentul scrierii programului, fapt ce nu permite utilizarea structurilor de date de tipul `array[ 1..m ] of Arbore`. Pentru a depăși acest inconvenient, tabloul respectiv poate fi înlocuit cu o listă uni- sau bidirecțională. Elaborați subprogramele necesare pentru crearea și prelucrarea arborilor multicăi de ordin arbitrar.



## 6.12. TIPUL DE DATE POINTER

Acest paragraf se referă în întregime la implementarea Turbo PASCAL.

**Mulțimea de valori** ale tipului predefinit de date pointer (indicator) constă din adrese și valoarea specială **nil**. Însă, spre deosebire de tipurile de date referință adresele cărora identifică numai variabilele dinamice ce aparțin tipului de bază, valorile de tip pointer pot identifica variabile dinamice de orice tip. Evident, valoarea **nil** nu identifică nici o variabilă dinamică. Prin convenție, tipul de date pointer este **compatibil** cu orice tip de date referință.

**Operațiile** care se pot face cu valori de tipul de date pointer sînt = și <>. Valorile de acest tip nu pot fi citite de la tastatură și afișate pe ecran.

O variabilă de tip pointer se introduce printr-o declarație de forma:

```
var p : pointer;
```

Întrucît astfel de declarații nu conțin informații despre tipul de bază, tipul variabilei dinamice  $p^{\wedge}$  este necunoscut. Prin urmare, variabilele de tip pointer nu pot fi dereperate, iar scrierea caracterului  $^{\wedge}$  după astfel de variabile constituie o eroare.

Programul ce urmează ilustrează utilizarea variabilelor de tip pointer pentru memorarea temporară a valorilor variabilelor de tip referință.

```
Program P134;
 { Tipul de date pointer }
var p : pointer;
 i, j : $^{\wedge}$ integer;
 x, y : $^{\wedge}$ real;
 r, s : $^{\wedge}$ string;
begin
 { p va identifica o variabilă dinamică de tipul
 integer }
 new(i); $i^{\wedge}:=1$;
 p:=i;
 new(i); $i^{\wedge}:=2$;
 j:=p;
 writeln('j^=', j^); { se afișează 1 }
 { p va identifica o variabilă dinamică de tipul
 real }
 new(x); $x^{\wedge}:=1$;
 p:=x;
 new(x); $x^{\wedge}:=2$;
 y:=p;
 writeln('y^=', y^); { se afișează 1.0000000000E+00 }
```

```

{ p va identifica o variabilă dinamică de tipul
 string }
new(r); r^:='AAA';
p:=r;
new(r); r^:='BBB';
s:=p;
writeln('s^=', s^); { se afișează AAA }
readln;
end.

```

Domeniul principal de utilizare a variabilelor de tip *pointer* este gestionarea memoriei interne a calculatorului. În Turbo PASCAL alocarea variabilelor dinamice se execută într-o zonă specială a memoriei interne numită **heap** (grămadă). Adresa de început a *heap*-ului, numită **adresa de bază**, este depusă în variabila predefinită de tip *pointer* *HeapOrg*. Variabila de tip *pointer* *HeapPtr* conține adresa primei locații libere, numită **vîrf** *heap*-ului (fig. 6.19).

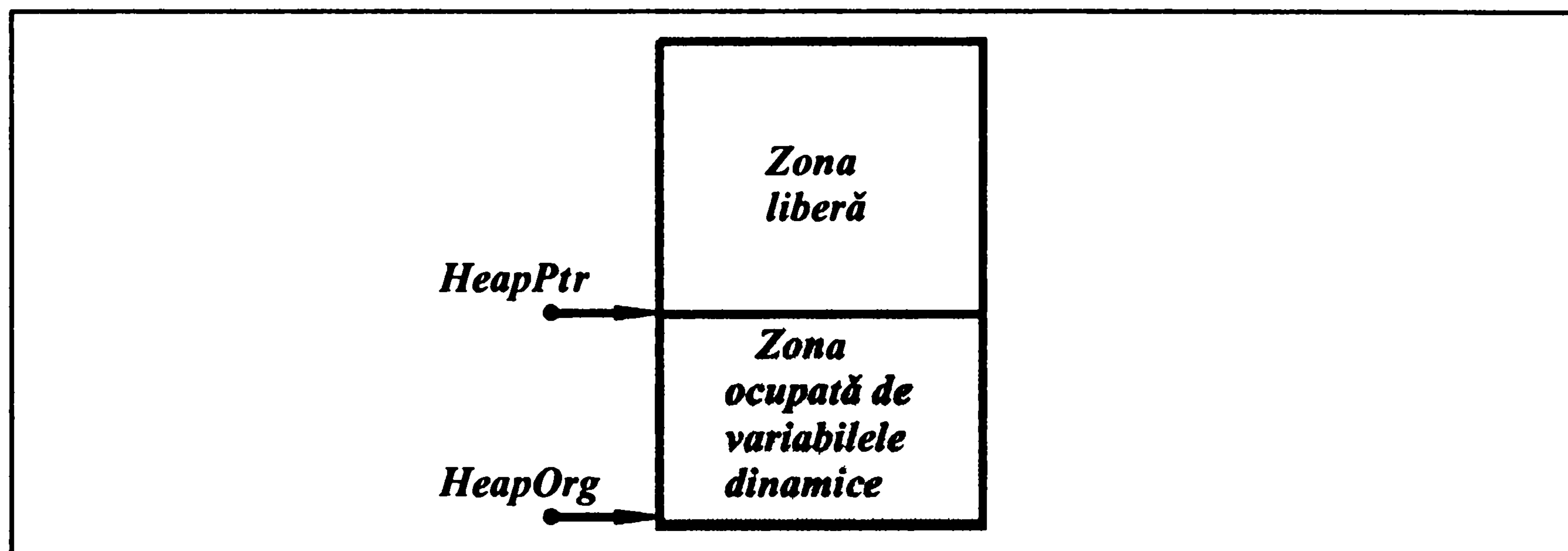


Fig. 6.19. Structura *heap*-ului

Variabilele dinamice sînt create și depuse în *heap* de procedura *new*. Ori de cîte ori în vîrf *heap*-ului se creează o variabilă dinamică conținutul variabilei *HeapPtr* este actualizat: valoarea curentă este incrementată cu dimensiunea spațiului de memorie necesar variabilei dinamice.

Memoria ocupată în *heap* de o variabilă dinamică se eliberează printr-un apel al procedurii *dispose*. Dimensiunea spațiului ce se eliberează depinde de tipul variabilei dinamice.

Ordinea de apelare a procedurii *dispose* nu coincide în general cu ordinea creării variabilelor dinamice de către procedura *new*. În consecință, în *heap* pot apărea goluri. Golurile apărute pot fi refolosite de procedura *new*, dacă variabila dinamică în curs de creare “încapă” în spațiul respectiv.

Eliberarea memoriei ocupate de o structură dinamică de date poate fi efectuată apelînd procedura *dispose* pentru fiecare componentă. Întrucît în program sînt cunoscute numai adresele componentelor pri-



vilegiate, de regulă baza și vârful listei, rădăcina arborelui etc., căutarea celorlalte componente cade în sarcina programatorului. Mai mult decît atît, ordinea de apelare a procedurii `dispose` trebuie să asigure păstrarea legăturilor către componentele care încă nu au fost distruse. În caz contrar, componentele respective nu mai sînt referite de nici un indicator de adresă și devin inaccesibile. Prin urmare, utilizarea procedurii `dispose` pentru eliberarea memoriei ocupate de structuri complexe de date este greoaie și inefficientă. Acest inconvenient poate fi depășit cu ajutorul procedurilor predefinite `mark` și `release`.

Apelul procedurii `mark` are forma:

```
mark(p)
```

unde `p` este o variabilă de tip `pointer`. Procedura memorează adresa vârfului din `HeapPtr` în variabila `p`.

Apelul procedurii `release` are forma:

```
release(p)
```

Această procedură reface adresa vârfului în starea înregistrată anterior cu procedura `mark`: valoarea conținută în variabila de tip `pointer` `p` este depusă în indicatorul `HeapPtr`.

Zona de memorie destinată alocării variabilelor dinamice poate fi gestionată cu ajutorul algoritmului ce urmează:

- 1) se memorează adresa vârfului cu procedura `mark`;
- 2) se creează variabilele dinamice cu procedura `new`;
- 3) se utilizează variabilele dinamice create;
- 4) cînd variabilele dinamice nu mai sînt necesare, spațiul ocupat din *heap* este eliberat cu procedura `release`.

*Exemplu:*

Se consideră următoarele declarații:

```
var i, j, k, m, n : ^integer;
 p : pointer;
```

Să presupunem că sînt executate instrucțiunile:

```
new(i); i^:=1;
new(j); j^:=2;
mark(p);
new(k); k^:=3;
new(m); m^:=4;
new(n); n^:=5;
```

Starea *heap*-ului este prezentată în *fig.6.20a*. Instrucțiunea `mark(p)` a memorat în variabila de tip `pointer` `p` valoarea actuală din `HeapPtr` înainte de crearea variabilei dinamice `k^`.

Dacă acum se execută instrucțiunea



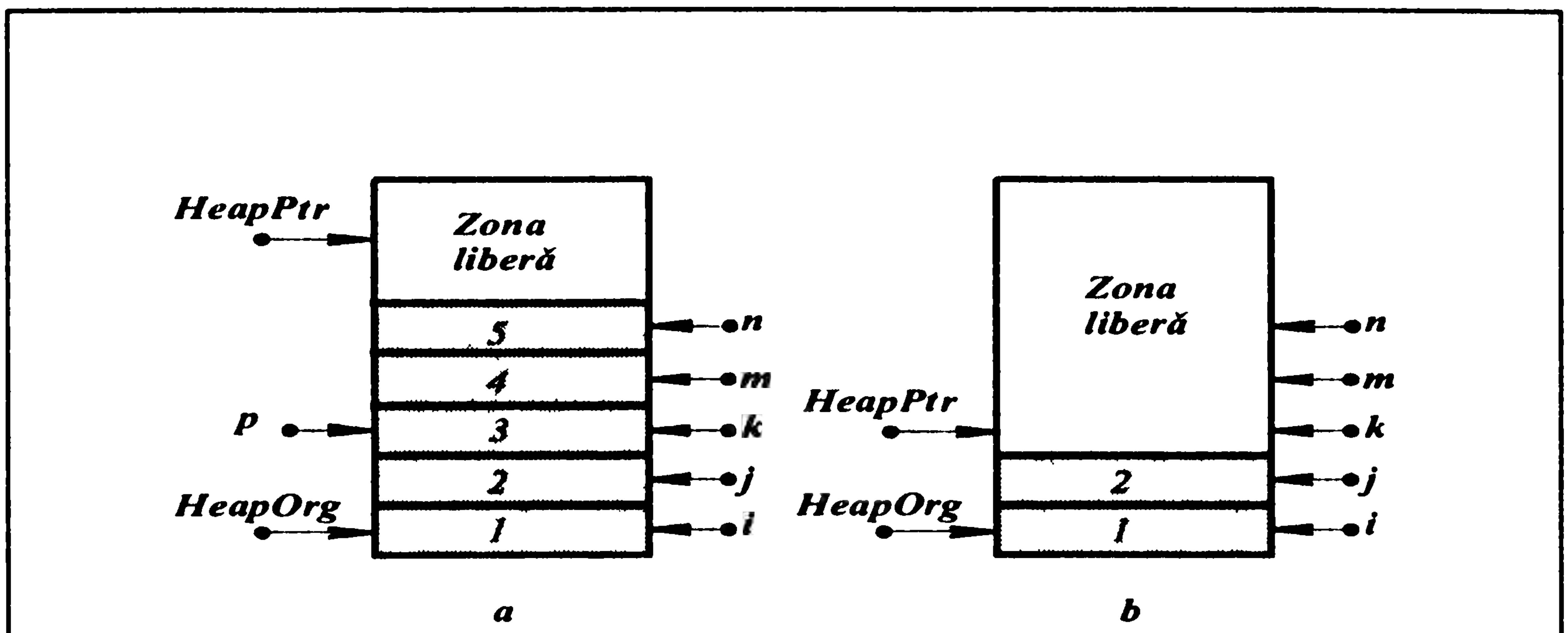


Fig. 6.20. Starea *heap*-ului  
pînă (a) și după (b) executarea instrucțiunii `release(p)`

`release(p)`

memoria ocupată de variabilele dinamice create după apelul procedurii `mark`, și anume,  $k^{\wedge}$ ,  $m^{\wedge}$  și  $n^{\wedge}$ , va fi eliberată (fig. 6.20b).

Deoarece variabila predefinită `HeapOrg` reține adresa de bază a *heap*-ului, tot spațiul destinat alocării variabilelor dinamice poate fi eliberat cu ajutorul instrucțiunii

`release(HeapOrg)`

Programul ce urmează ilustrează utilizarea procedurilor `mark` și `release`.

```

Program P135;
 { Gestionarea memoriei interne }
 type Lista = ^Celula;
 Celula = record
 Info : string;
 Urm : Lista
 end;
 Stiva = Lista;
 Arbore = ^Nod;
 Nod = record
 Info : string;
 Stg, Dr : Arbore
 end;
 var L : Lista;
 S : Stiva;
 T : Arbore;
 p : pointer;

 function Lst : Lista;
 { crearea listei unidirecționale }

```

```

var R : Lista;
 s : string;
begin
 write('Info='); readln(s);
 if s='' then Lst:=nil
 else
 begin
 new(R);
 R^.Info:=s;
 R^.Urm:=Lst;
 Lst:=R;
 end;
end; { Lst }

procedure AfisLst(L : Lista);
 { afişarea listei }
begin
 if L<>nil then
 begin
 writeln(L^.Info);
 AfisLst(L^.Urm);
 end;
end; { AfisLst }

procedure Stv(var S : Stiva);
 { crearea unei stive }
var R : Stiva;
 st : string;
begin
 S:=nil;
 write('Info='); readln(st);
 while st<>'' do
 begin
 new(R);
 R^.Info:=st;
 R^.Urm:=S;
 S:=R;
 write('Info='); readln(st);
 end;
end; { Stv }

function Arb : Arbore;
 { crearea arborelui binar }
var R : Arbore;
 s : string;
begin
 readln(s);
 if s='' then Arb:=nil
 else begin
 new(R);

```

```

 R^.Info:=s;
 write('Dați descendentul stîng');
 writeln(' al nodului ', s, ':');
 R^.Stg:=Arb;
 write('Dați descendentul drept');
 writeln(' al nodului ', s, ':');
 R^.Dr:=Arb;
 Arb:=R;
 end;
end; { Arb }

procedure AfisArb(T : Arbore; nivel : integer);
{ afișarea arborelui binar }
var i : integer;
begin
 if T<>nil then
 begin
 AfisArb(T^.Stg, nivel+1);
 for i:=1 to nivel do write(' ');
 writeln(T^.Info);
 AfisArb(T^.Dr, nivel+1);
 end;
 end; { AfisArb }
begin
 writeln('Dați lista:');
 L:=Lst;
 writeln('Lista creată:');
 AfisLst(L);
 mark(p); { p reține adresa din HeapPtr }
 writeln('Dați rădăcina:');
 T:=Arb;
 writeln('Arborele creat:');
 AfisArb(T, 0);
 release(p); { eliberarea memoriei ocupate de arbore }
 writeln('Dați stiva:');
 Stv(S);
 writeln('Stiva creată');
 AfisLst(S);
 release(HeapOrg); { eliberarea memoriei ocupate
 de listă și stivă }

 readln;
end.

```

Subliniem faptul că în implementările actuale procedurile `dispose` și `release` nu atribuie valoarea **nil** indicatorilor de adresă variabilele dinamice ale cărora au fost distruse (fig. 6.20b). Întrucît memoria eliberată este refolosită, atribuirile efectuate asupra variabilelor distruse pot altera valorile variabilelor dinamice nou-create.



## Întrebări și exerciții

- ❶ Care este mulțimea de valori ale tipului de date `pointer`? Ce operații pot fi efectuate cu aceste valori?
- ❷ Comentați următorul program:

```
Program P136;
 { Eroare }
var i : ^integer;
 j, k : integer;
 p : pointer;
begin
 new(i); i^:=1;
 p:=i;
 new(i); i^:=2;
 j:=i^; k:=p^;
 writeln('j+k=', j+k);
end.
```

- ❸ Care este domeniul de utilizare a variabilelor de tip `pointer`?
- ❹ Este oare *heap*-ul o structură de date de tip stivă? Argumentați răspunsul.
- ❺ Lansați în execuție programele ce urmează. Explicați rezultatele afișate pe ecran.

```
Program P137;
var i, j, k, m, n : ^integer;
 p : pointer;
begin
 { crearea variabilelor i^, j^, k^ }
 new(i); new(j); new(k);
 i^:=1; j^:=2; k^:=3;
 p:=j; { p reține adresa din j }
 { distrugerea variabilei j^ și crearea variabilei m^ }
 dispose(j); new(m); m^:=4;
 j:=p; { refacerea adresei din j }
 writeln('i^=', i^, ' j^=', j^, ' k^=', k^);
 { distrugerea variabilei m^ și crearea variabilei n^ }
 dispose(m); new(n); n^:=5;
 writeln('i^=', i^, ' j^=', j^, ' k^=', k^);
 readln;
end.
```

```
Program P138;
var i, j, k, m : ^integer;
begin
 { crearea variabilelor i^, j^ }
 new(i); new(j);
 i^:=1; j^:=2;

 { eliberarea memoriei heap-ului }
 release(HeapOrg);
```

```
{ crearea variabilelor k^ și m^ }
new(k); new(m);
k^:=1; m^:=2;
writeln('k^=', k^, ' m^=', m^);
i^:=3; j^:=4;
writeln('k^=', k^, ' m^=', m^);
readln;
end.
```

⑥ Scrieți o procedură care eliberează memoria ocupată de:

- a) o listă unidirecțională;
- b) un arbore binar;
- c) un arbore multicăi.

Memoria trebuie eliberată apelând procedura `dispose` pentru fiecare componentă a structurii dinamice de date.

⑦ Scrieți un program în care se creează mai întâi o coadă, iar apoi un arbore multicăi. Spațiul de memorie eliberat după distrugerea cozii trebuie refolosit pentru alocarea arborelui.

# METODE DE ELABORARE A PRODUSELOR PROGRAM

## 7.1. PROGRAMAREA MODULARĂ

Programarea modulară urmărește reducerea complexității programelor mari prin descompunerea acestora în module.

**Modulul** este un produs program format din descrieri de date și subprograme destinate prelucrării acestora. Modulele pot fi scrise independent și compilate separat înainte de a fi încorporate în programul în curs de elaborare. Menționăm că pentru program se mai utilizează și denumirea de **modul principal**.

Limbajul-standard nu prevede mijloace pentru programarea modulară. Se consideră că programele PASCAL sînt entități monolit care trebuie compilate împreună cu subprogramele pe care, eventual, le conțin. Acest lucru devine incomod în cazul programelor mari care pot include zeci și chiar sute de subprograme.

În versiunea Turbo PASCAL modulele sînt implementate prin **unități de program**. Forma generală a unei **unități de program** este:

```
unit <Nume>;
interface
[uses <Nume>{ , <Nume>} ;]
[<Constante>]
[<Tipuri>]
[<Variabile>]
[{ <Antet funcție>; | <Antet procedură>; }]

implementation
[uses <Nume> { , <Nume>} ;]
[<Etichete>]
[<Constante>]
[<Tipuri>]
[<Variabile>]
[<Subprograme>]
[{ function <Identificator>;
 <Corp>; |
 procedure <Identificator>;
```



```

<Corp>;}]
[begin
[<Instrucțiune> { ; <Instrucțiune > }]]
end.

```

În esență, o unitate de program constă din trei secțiuni: de interfață, de implementare și de inițializare.

**Secțiunea de interfață** începe cu cuvântul-cheie **interface**. Aici se declară constantele, tipurile, variabilele și subprogramele exportate de unitate. Aceste elemente pot fi referite de orice modul care utilizează direct sau prin tranzitivitate unitatea respectivă. Menționăm că în secțiunea de interfață apar doar antetele funcțiilor și procedurilor exportate. Dacă unitatea actuală utilizează alte unități, numele acestora sînt specificate în clauza **uses**.

**Secțiunea de implementare** începe cu cuvântul-cheie **implementation**. Această secțiune conține declarații locale de etichete, constante, tipuri, variabile și subprograme. Elementele definite aici sînt „ascunse” și nu pot fi referite de modulele care utilizează unitatea actuală. După declarațiile locale urmează corpul procedurilor și funcțiilor, ale căror antete au fost definite în secțiunea de interfață. Fiecare subprogram specificat în interfață trebuie să aibă un corp. După cuvântul-cheie **function** sau **procedure** se scrie doar numele subprogramului. Ca și în cazul declarațiilor anticipate, nu este necesară descrierea listei de parametri și a valorii returnate.

**Secțiunea de inițializare** începe, dacă există, cu cuvântul-cheie **begin**. Secțiunea constă dintr-o secvență de instrucțiuni și servește pentru atribuirea valorilor inițiale variabilelor definite în secțiunea de interfață. Dacă un program utilizează mai multe unități, execuția programului este precedată de execuția secțiunilor de inițializare în ordinea în care aceste unități apar în clauza **uses** din program.

*Exemplu:*

```

Unit U1;
 { Prelucrarea vectorilor }
interface

 const nmax=100;
 type Vector=array[1..nmax] of real;
 var n : 1..nmax;
 function sum(V : Vector) : real;
 function min(V : Vector) : real;
 function max(V : Vector) : real;
 procedure Citire(var V : Vector);
 procedure Afisare(V : Vector);

implementation

```

```

var i : 1..nmax;
 s : real;

function sum;
begin
 s:=0;
 for i:=1 to n do s:=s+V[i] ;
 sum:=s;
end; { sum }

function min;
begin
 s:=V [1] ;
 for i:=2 to n do
 if s>V[i] then s:=V[i] ;
 min:=s;
end; { min }

function max;
begin
 s:=V[1] ;
 for i:=2 to n do
 if s<V[i] then s:=V[i] ;
 max:=s;
end; { max }

procedure Citire;
begin
 for i:=1 to n do readln(V[i]);
end; { Citire }

procedure Afisare;
begin
 for i:=1 to n do writeln(V[i]);
end; { Afisare }

begin
 write('n='); readln(n);
end.

```

Unitatea U1 exportă constanta nmax, tipul Vector, variabila n, funcțiile sum, min, max, procedurile Citire și Afisare. Valoarea inițială a variabilei n este citită de la tastatură. Elementele în studiu pot fi referite în orice program ce conține clauza **uses** U1.

*Exemplu:*

|                                                                                |
|--------------------------------------------------------------------------------|
| <pre> <b>Program</b> P139;   { Utilizarea unității U1 } <b>uses</b> U1; </pre> |
|--------------------------------------------------------------------------------|

```

var A : Vector;
begin
 writeln('Dați un vector:');
 Citire(A);
 writeln('Ați introdus:');
 Afisare(A);
 writeln('sum=', sum(A));
 writeln('min=', min(A));
 writeln('max=', max(A));
 readln;
end.

```

**Domeniile de vizibilitate** ale declarațiilor din unitățile de program se stabilesc conform regulilor ce urmează.

1. Declarațiile din secțiunea de implementare sînt vizibile numai în unitatea actuală.

2. Declarațiile din secțiunea de interfață sînt vizibile în:

- unitatea actuală;
- modulele care utilizează direct unitatea actuală;
- modulele care utilizează unitatea actuală prin tranzitivitate.

Referirea oricărui identificator *id* declarat într-o unitate *v* utilizată prin tranzitivitate se face prin *v.id*.

3. Dacă unul și același identificator este declarat în mai multe module, este luată în considerație declarația cea mai recentă.

*Exemplu:*

```

Program P140;
uses U2;
var x : integer;
begin
 x:=4;
 writeln('Programul P140:');
 writeln('n=', U3.n);
 writeln('m=', m);
 writeln('x=', x);
 readln;
end.

```

```

Unit U2;
interface
 uses U3;
 var m : integer;
 x : real;
implementation
begin
 writeln('Unitatea U2:');
 m:=2;
 writeln(' m=', m);

```



```

 x:=3.0;
 writeln(' x=', x);
end.

Unit U3;
interface
 var n : integer;
implementation
begin
 writeln('Unitatea U3:');
 n:=1;
 writeln('n=', n);
end.

```

În programul P140 unitatea U2 este utilizată direct, iar unitatea U3 prin tranzitivitate. Variabila *n* din modulul U3 este referită prin *U3.n*. Identificatorul *x* apare în declarațiile **var** *x*: *real* din unitatea U2 și **var** *x*: *integer* din programul P140. Compilatorul ia în considerație ultima declarație.

Unitățile de program se clasifică în unitățile-standard, livrate o dată cu compilatorul *Turbo PASCAL*, și unitățile scrise de utilizator. În continuare prezentăm o caracteristică succintă a unităților-standard frecvent utilizate.

**System** — conține toate subprogramele predefinite din Turbo PASCAL. Unitatea în studiu se încorporează automat în toate programele, fără a fi necesară clauza **uses**.

**Crt** — permite utilizarea funcțiilor și procedurilor referitoare la lucrul cu ecranul în mod text, precum și cu tastatura și difuzorul. Accesibilitatea subprogramelor se realizează prin clauza **uses crt**.

**Graph** — implementează subprogramele destinate unor prelucrări grafice: definiri de ferestre și pagini, definiri de culori și palete, desenarea arcurilor, cercurilor, poligoanelor și altor figuri, salvarea imaginilor etc. Serviciile unității de program pot fi accesate prin clauza **uses graph**.

**Printer** — asigură redirectarea operațiilor de scriere în fișierul text cu numele *lst* la imprimantă. Utilizând unitatea în studiu, programatorul nu mai trebuie să declare, să deschidă și să închidă acest fișier. Serviciile unității **Printer** devin accesibile unui program sau unei unități de program prin specificarea clausei **uses printer**.

Destinația și modul de utilizare a constantelor, tipurilor de date, variabilelor, funcțiilor și procedurilor din unitățile-standard este inclusă în ghidurile de utilizare și sistemele de asistență *Turbo PASCAL's Online Help*.

Elaborând propriile unități de program, orice utilizator își poate crea biblioteci de subprograme ce descriu algoritmi din diverse

domenii: rezolvarea ecuațiilor, calcule statistice, procesarea textelor, crearea și prelucrarea structurilor dinamice de date etc. Divizarea unui program mare în module ușurează activitatea de elaborare a produselor program în echipă. În astfel de cazuri fiecare programator scrie, testează și documentează câteva module relativ simple, ceea ce contribuie la îmbunătățirea produsului program rezultat.

## Întrebări și exerciții

- ❶ Care sînt avantajele programării modulare? Prevede oare limbajul-standard mijloace pentru programarea modulară?
- ❷ Care este forma-standard a unei unități de program? Precizați structura și destinația secțiunilor de interfață, de implementare și de inițializare.
- ❸ Cum se determină domeniile de vizibilitate ale declarațiilor din unitățile de program?
- ❹ Precizați ce va afișa pe ecran programul ce urmează.

```
Program P141;
uses U4;
var s : string;
begin
 s:='BBB';
 writeln('U5.k=', U5.k);
 writeln('U5.m=', U5.m);
 writeln('U5.s=', U5.s);
 writeln('U4.m=', U4.m);
 writeln('U4.s=', U4.s);
 writeln('m=', m);
 writeln('s=', s);
 readln;
end.
```

```
Unit U4;
interface
uses U5;
var m : real;
 s : char;
implementation
begin
 m:=4.0;
 s:='A';
end.

Unit U5;
interface
var k, m : integer;
 s : real;
implementation
begin
 k:=1;
 m:=2;
```



```
s:=3.0;
end.
```

⑤ Comentați programul:

```
Program P142;
 { Eroare }
uses U6;
begin
 writeln('k=', k);
 writeln('m=', m);
 readln;
end.
```

```
Unit U6;
interface
 var k : integer;
implementation
 var m : integer;
begin
 k:=1;
 m:=2;
end.
```

⑥ Completați unitatea de program U1 din paragraful în studiu cu un subprogram care:

- a) returnează media aritmetică a componentelor unui vector;
- b) aranjează componentele în ordine crescătoare;
- c) returnează produsul componentelor unui vector;
- d) returnează numărul componentelor pozitive;
- e) aranjează componentele în ordine descrescătoare.

⑦ Scrieți o unitate de program care oferă descrieri de date și subprograme pentru prelucrarea matricelor.

⑧ Numerele întregi  $n$ ,  $n \leq 10^{254}$ , pot fi reprezentate în calculator prin șiruri formate din caracterele '+', '-', '0', '1', '2', ..., '9'. Elaborați o unitate de program care conține funcțiile și procedurile necesare pentru efectuarea următoarelor operații:

- a) citirea numerelor de la tastatură;
- b) afișarea numerelor pe ecran;
- c) +, -, \*, mod, div;
- d) calcularea factorialului;
- e) citirea și scrierea numerelor în fișiere secvențiale.

⑨ Șirurile de caractere de lungime  $n$ ,  $n \leq 500$ , pot fi reprezentate în programele *Turbo PASCAL* prin variabile de tipul

```
type lungime = 0..500;
 SirDeCaractere = record
 n: lungime;
 s: array[1..500] of char
 end;
```



Elaborați o unitate de program care conține funcțiile și procedurile necesare pentru efectuarea următoarelor operații:

- a) citirea șirurilor de la tastatură;
- b) afișarea șirurilor pe ecran;
- c) concatenarea șirurilor;
- d) compararea lexicografică;
- e) calcularea lungimii unui șir.

⑩ Elaborați o unitate de program pentru prelucrarea:

- a) listelor unidirecționale;
- b) listelor bidirecționale;
- c) cozilor;
- d) stivelor;
- e) arborilor binari;
- f) arborilor binari de căutare;
- g) arborilor multicai.

Utilizați în acest scop declarațiile de tipuri, funcții și proceduri din capitolul 6.

⑪ Găsiți în sistemul de asistență *Turbo PASCAL's Online Help* descrierea unităților-standard instalate pe calculatorul D-voastră. Afișați pe ecran textul fiecărei unități, determinați destinația și modul de utilizare a funcțiilor și procedurilor respective.

## 7.2. TESTAREA ȘI DEPANAREA PROGRAMELOR

Un program este **corect** dacă :

- a) după lansarea în execuție procesul de calcul se termină;
- b) rezultatele obținute reprezintă o soluție a problemei pentru rezolvarea căreia a fost scris programul.

În caz contrar programul conține **erori**.

Asigurarea corectitudinii unui program presupune execuția sa pentru fiecare combinație posibilă de valori ale datelor de intrare. În majoritatea cazurilor acest lucru este imposibil, deoarece domeniul de valori al datelor de intrare este, practic, infinit, iar soluțiile respective sînt necunoscute.

**Testarea** este o etapă în elaborarea programelor ce are drept scop eliminarea erorilor. Ea se realizează executînd programul cu anumite seturi de date de intrare numite **date de testare** sau, mai simplu, **teste**. În funcție de modul de selecție a datelor de testare, deosebim :

- testarea funcțională sau metoda cutiei negre ;
- testarea structurală sau metoda cutiei transparente.

Amintim că termenul „*cutie neagră*” este folosit pentru un sistem, structura internă a căruia este necunoscută.

În cazul **testării funcționale** datele de testare sînt astfel concepute, încît să se asigure că fiecare funcție a programului este pe deplin

realizată. Programul este văzut ca o cutie neagră, a cărei funcționare este determinată prin introducerea unor date și analiza rezultatelor obținute. Selectarea datelor de intrare depinde, în mare măsură, de îndemânarea și experiența celui care efectuează testarea. În mod obișnuit, se selectează **valori tipice** și **valori netipice** din domeniul datelor de intrare.

*Exemplu.* Se consideră programul P143. Textul programului nu este deocamdată prezentat pentru a sublinia faptul că în metoda testării funcționale el nu este necesar. Programul realizează următoarele funcții :

- citește de la tastatură un șir de numere reale;
- afișează pe ecran media aritmetică a numerelor pozitive din șir.

Evident, domeniul datelor de intrare este format din șiruri de numere reale.

Datele de testare vor include :

a) valorile netipice:

- șir vid;
- șir ce nu conține numere pozitive;
- șir ce conține un singur număr pozitiv;

b) valorile tipice:

- șir cu două numere pozitive;
- șir cu trei sau mai multe numere pozitive.

În testarea structurală testele sînt elaborate examinînd structura programului: declarațiile de date, proceduri și funcții, instrucțiunile simple, instrucțiunile structurate etc. Datele de testare vor asigura:

a) execuția fiecărei instrucțiuni simple (atribuiri, apeluri de proceduri, salturi **goto**);

b) execuția fiecărui ciclu **for** de zero, unu și de mai multe ori;

c) execuția fiecărei instrucțiuni **if**, **repeat**, **while** pentru valorile true și false ale expresiilor booleene de control;

d) execuția fiecărui caz din componența instrucțiunilor **case**.

*Exemplu.* Prezintă textul programului care calculează media aritmetică a numerelor pozitive dintr-un șir:

```
Program P143;
{ Media numerelor pozitive dintr-un șir }
var n, k : integer;
 x, s : real;
begin
 n:=0;
 k:=0;
 s:=0;
 writeln('Dați un șir de numere reale:');
 while not eof do
 begin
```



```

 readln(x);
 n:=n+1;
 if x>0 then
 begin
 k:=k+1;
 s:=s+x;
 end;
 end; { while }
 if n=0 then writeln('Șir vid')
 else if k=0 then
 writeln('Șirul nu conține numere pozitive')
 else writeln('Media=', s/k);
 readln;
end.

```

Testul trebuie să asigure execuția instrucțiunilor **while** și **if** pentru valorile true și false ale expresiilor booleene **not eof**,  $x > 0$ ,  $n = 0$  și  $k = 0$ . Prin urmare, datele de testare vor include:

- un șir vid (**not eof**=false,  $n = 0$ );
- un șir nevid (**not eof**=true,  $n \neq 0$ );
- un șir ce conține cel puțin un număr pozitiv ( $x > 0$ ,  $k \neq 0$ );
- un șir nevid ce nu conține numere pozitive ( $x \leq 0$ ,  $k = 0$ ).

**Metoda cutiei transparente** poate fi utilizată independent sau împreună cu metoda cutiei negre pentru a îmbunătăți un test deja obținut. De exemplu, în cazul programului P143 șirul ce conține cel puțin un număr pozitiv poate fi înlocuit cu trei șiruri ce conțin, respectiv unul, două, trei sau mai multe numere pozitive. Aceste date vor asigura execuția instrucțiunilor  $k := k + 1$ ,  $s := s + x$  și calcularea expresiei  $s/k$  pentru valorile tipice și valorile netipice ale variabilelor  $k$  și  $s$ .

Dificultățile în aplicarea testării structurale sînt legate de prezența instrucțiunilor de decizie (**if**, **case**), a celor iterative (**for**, **while**, **repeat**) sau a celei de transfer (**goto**). Acestea determină apariția unui număr foarte mare de combinații, în care instrucțiunile de atribuire și apelurile de proceduri pot fi executate.

**Depanarea programului** constă în localizarea zonelor din program care au condus la apariția unei erori, identificarea cauzelor erorii și corectarea acesteia. Depanarea poate fi făcută static (după executarea programului) și dinamic (în timpul executării).

În metoda **depanării statice** cauzele erorii se stabilesc, analizînd rezultatele derulării programului și mesajele sistemului de operare. Pentru a facilita procesul de depanare, în program temporar se includ instrucțiuni care afișează pe ecran valorile intermediare ale variabilelor-cheie.

În metoda **depanării dinamice** localizarea erorilor se face urmărind executarea programului la nivel de instrucțiuni. Implementările



actuale ale limbajului permit efectuarea următoarelor operații de depanare dinamică:

- execuția pas cu pas a programului;
- observarea valorilor unor expresii specificate;
- crearea și eliminarea unor puncte de suspendare a executării;
- modificarea valorilor unor variabile;
- trasarea apelurilor de funcții și proceduri;
- tratarea erorilor de intrare–ieșire, a erorilor de depășire etc.

Descrierea detaliată a operațiilor în studiu este inclusă în sistemul de asistență *Turbo PASCAL's Online Help*.

Eficiența depanării depinde de modul în care este scris și testat programul, calitatea mesajelor de eroare generate de calculator și tipul erorii. De regulă, un test care semnalează prezența unei erori este urmat de alte texte organizate în așa fel, încât să izoleze eroarea și să furnizeze informații pentru corectarea ei.

S-a constatat că testarea și depanarea ocupă mai mult de jumătate din perioada de timp necesară realizării unui produs program. Complexitatea acestor procese poate fi redusă prin divizarea programelor mari în subprograme sau module și aplicarea metodelor programării structurate.

Subliniem faptul că testarea programelor este un mijloc eficient de a **depista erorile**, însă nu și un mijloc de a demonstra absența lor. Cu toate că testarea nu demonstrează corectitudinea programului, ea este deocamdată singura metodă practică de certificare a produselor program. În prezent se elaborează metode de verificare bazate pe demonstrarea formală a corectitudinii programului, însă rezultatele cunoscute în această direcție nu sînt aplicabile programelor complexe.

## Întrebări și exerciții

- ❶ Când un program PASCAL este corect? Cum poate fi asigurată corectitudinea unui program?
- ❷ Cum se selectează datele de intrare în metoda testării funcționale?
- ❸ Elaborați un test funcțional pentru programul P124 din paragraful 6.4. Programul realizează următoarele funcții:
  - creează o listă unidirecțională;
  - afișează lista pe ecran;
  - include un anumit element în listă;
  - exclude din listă elementul specificat de utilizator.
- ❹ Precizați funcțiile realizate de programele ce urmează și elaborați testele funcționale:
  - a) P49, P53, P54 și P69 din capitolul 3;
  - b) P78, P82, P85, P89 și P90 din capitolul 4;
  - c) P127, P128 și P132 din capitolul 6.

- ⑤ Cum se selectează datele de intrare în metoda testării structurale?
- ⑥ Elaborați teste structurale pentru programele ce urmează:
  - a) P49, P53, P60, P66 și P68 din capitolul 3;
  - b) P78, P82, P85, P93 și P94 din capitolul 4;
  - c) P127, P128 și P130 din capitolul 6.
- ⑦ Care este diferența dintre *depanarea statică* și *depanarea dinamică*?
- ⑧ Găsiți în sistemul de asistență *Turbo PASCAL's Online Help* descrierea operațiilor de depanare dinamică. Efectuați aceste operații pentru programele elaborate de D-voastră.

### 7.3. ELEMENTE DE PROGRAMARE STRUCTURATĂ

Încă din primii ani de activitate în domeniul prelucrărilor de date s-a constatat că testarea, depanarea și modificarea programelor necesită un mare volum de muncă. Mai mult decât atât, programele complexe ce conțin sute și mii de instrucțiuni devin greu accesibile chiar și pentru autorii lor.

**Programarea structurată** reprezintă un stil, o manieră de concepere a programelor potrivit unor reguli bine stabilite, bazate pe teorema de structură. Conform **teoremei de structură**, orice algoritm poate fi reprezentat ca o combinație a trei structuri de control:

- secvența (succesiune de două sau mai multe atribuiri și/sau apeluri de proceduri);
- decizia (**if... then... sau if... then... else...**);
- ciclul cu test inițial (**while... do...**).

Programarea structurată admite și utilizarea altor structuri de control, cum sînt:

- selecția (**case... of...**);
- ciclul cu test final (**repeat... until...**);
- ciclul cu contor (**for... do...**).

Regulile de bază ale programării structurate sînt:

1. Structura oricărui program sau subprogram va fi concepută ca o combinație a structurilor de control admise: secvența, decizia, selecția, ciclul.

2. Structura datelor utilizate în program trebuie să corespundă specificului problemelor rezolvate.

3. Lungimea maximă a unei funcții sau proceduri este de 50–100 linii. Folosirea variabilelor globale nu este încurajată.

4. Identificatorii folosiți pentru constante, tipuri, variabile, funcții, proceduri și unități de program trebuie să fie cât mai sugestivi.



5. Claritatea textului trebuie asigurată prin inserarea comentariilor și alinierea textului în conformitate cu structura logică și sintactică a instrucțiunilor.

6. Operațiile de intrare-ieșire vor fi localizate în subprograme separate. Corectitudinea datelor de intrare se verifică imediat după citirea lor.

7. Încuibarea instrucțiunilor **if** mai mult de trei ori trebuie evitată prin folosirea instrucțiunilor **case**.

Programele obținute conform regulilor în studiu sînt testabile, clare, ordonate, fără salturi și reveniri. Menționăm că, conform teoremei de structură, orice program poate fi scris fără a utiliza instrucțiunea **goto**. Totuși unii autori admit utilizarea acestei instrucțiuni cu condiția ca ea să fie folosită la minimum, iar salturile să fie efectuate numai în jos.

### Întrebări și exerciții

- ❶ Care este justificarea teoretică a programării structurate?
- ❷ Precizați structurile de control necesare și suficiente pentru reprezentarea oricărui algoritm.
- ❸ Formulați regulile de bază ale programării structurate.
- ❹ Care sînt avantajele programării structurate?
- ❺ Corespund oare programele P124, P130 și P135 din capitolul 6 regulilor de bază ale programării structurate?
- ❻ Programul ce urmează afișează pe ecran toate reprezentările posibile ale numărului natural  $n$  ca sumă de numere naturale consecutive.

```
Program P144;
var a,i,l,s,n : integer;
 b : boolean;

begin
 write('n='); readln(n);
 b:=true;
 for i:=1 to ((n+1) div 2) do
 begin
 a:=i;
 s:=0;
 while (s<n) do
 begin
 s:=s+a;
 a:=a+1;
 end;
 if s=n then
 begin
 b:=false;
 write(n, '=', i);
 for l:=i+1 to (a-1) do write('+',l);
 writeln;
 end;
 end;
```



```
end;
if b then writeln('Reprezentări nu există');
readln;
end.
```

De exemplu, pentru  $n = 15$  obținem :

$$15 = 1 + 2 + 3 + 4 + 5;$$

$$15 = 4 + 5 + 6;$$

$$15 = 7 + 8.$$

Reprezentările în studiu se calculează prin metoda trierii, examinându-se șirurile :

1, 2, 3, ...,  $k$ ;

2, 3, ...,  $k$ ;

3, ...,  $k$

ș.a.m.d., unde  $k = (n + 1) \div 2$ . Termenii 1, 2, 3, ...,  $k$  se calculează cu ajutorul relației recurente  $a = a + 1$ .

Aliniați textul programului în conformitate cu structura logică și sintaxa fiecărei instrucțiuni.

## Anexa 1. VOCABULARUL LIMBAJULUI PASCAL

1.  $\langle \text{Literă} \rangle ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z$
2.  $\langle \text{Cifră} \rangle ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9$
3.  $\langle \text{Simbol special} \rangle ::= + | - | * | / | = | < | > | ] | [ | , | ( | ) | : | ; | ^ | . | @ | \{ | \} | \$ | \# | < = | = > | < > | := | . . | \langle \text{Cuvînt-cheie} \rangle | \langle \text{Simbol echivalent} \rangle$
4.  $\langle \text{Simbol echivalent} \rangle ::= ( * | * ) | ( . | . )$
5.  $\langle \text{Cuvînt-cheie} \rangle ::= \text{and} | \text{array} | \text{begin} | \text{case} | \text{const} | \text{div} | \text{do} | \text{downto} | \text{else} | \text{end} | \text{file} | \text{for} | \text{function} | \text{goto} | \text{if} | \text{in} | \text{label} | \text{mod} | \text{nil} | \text{not} | \text{of} | \text{or} | \text{packed} | \text{procedure} | \text{program} | \text{record} | \text{repeat} | \text{set} | \text{then} | \text{to} | \text{type} | \text{until} | \text{var} | \text{while} | \text{with}$
6.  $\langle \text{Identificator} \rangle ::= \langle \text{Literă} \rangle \{ \langle \text{Literă} \rangle | \langle \text{Cifră} \rangle \}$
7.  $\langle \text{Directivă} \rangle ::= \langle \text{Literă} \rangle \{ \langle \text{Literă} \rangle | \langle \text{Cifră} \rangle \}$
8.  $\langle \text{Întreg fără semn} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$
9.  $\langle \text{Semn} \rangle ::= + | -$
10.  $\langle \text{Număr întreg} \rangle ::= [ \langle \text{Semn} \rangle ] \langle \text{Întreg fără semn} \rangle$
11.  $\langle \text{Factor scală} \rangle ::= \langle \text{Număr întreg} \rangle$
12.  $\langle \text{Număr real} \rangle ::= \langle \text{Număr întreg} \rangle \text{e} \langle \text{Factor scală} \rangle | \langle \text{Număr întreg} \rangle . \langle \text{Întreg fără semn} \rangle [ \text{e} \langle \text{Factor scală} \rangle ]$
13.  $\langle \text{Număr} \rangle ::= \langle \text{Număr întreg} \rangle | \langle \text{Număr real} \rangle$
14.  $\langle \text{Șir de caractere} \rangle ::= ' \langle \text{Element șir} \rangle \{ \langle \text{Element șir} \rangle \} '$
15.  $\langle \text{Element șir} \rangle ::= '' | \langle \text{Orice caracter imprimabil} \rangle$
16.  $\langle \text{Etichetă} \rangle ::= \langle \text{Întreg fără semn} \rangle$
17.  $\langle \text{Comentariu} \rangle ::= ( * \langle \text{Orice secvență de caractere și sfârșit de linie neconținând acolade drepte} \rangle * )$

*Notă.* Simbolurile terminale { și } din formula (17) sînt redate prin simbolurile echivalente respectiv (\* și \*).

## Anexa 2. SINTAXA LIMBAJULUI PASCAL

1.  $\langle \text{Program} \rangle ::=$   
     $\langle \text{Antet program} \rangle$   
     $\langle \text{Corp} \rangle .$
2.  $\langle \text{Antet program} \rangle ::=$   
    **Program**  $\langle \text{Identificator} \rangle$   
     $[ ( \langle \text{Identificator} \rangle \{ , \langle \text{Identificator} \rangle \} ) ] ;$

3.  $\langle \text{Corp} \rangle ::= \langle \text{Declarații} \rangle$   
 $\quad \langle \text{Instrucțiune compusă} \rangle$
4.  $\langle \text{Declarații} \rangle ::=$   $[\langle \text{Etichete} \rangle]$   
 $\quad [\langle \text{Constante} \rangle]$   
 $\quad [\langle \text{Tipuri} \rangle]$   
 $\quad [\langle \text{Variabile} \rangle]$   
 $\quad [\langle \text{Subprograme} \rangle]$
5.  $\langle \text{Etichete} \rangle ::= \text{label } \langle \text{Etichetă} \rangle \{, \langle \text{Etichetă} \rangle \};$
6.  $\langle \text{Constante} \rangle ::=$   
 $\quad \text{const } \langle \text{Definiție constantă} \rangle; \{ \langle \text{Definiție constantă} \rangle; \}$
7.  $\langle \text{Definiție constantă} \rangle ::= \langle \text{Identificator} \rangle = \langle \text{Constantă} \rangle$
8.  $\langle \text{Constantă} \rangle ::=$   
 $\quad [+ \mid -] \langle \text{Număr fără semn} \rangle \mid$   
 $\quad [+ \mid -] \langle \text{Nume de constantă} \rangle \mid$   
 $\quad \langle \text{Șir de caractere} \rangle$
9.  $\langle \text{Tipuri} \rangle ::= \text{type } \langle \text{Definiție tip} \rangle; \{ \langle \text{Definiție tip} \rangle ; \}$
10.  $\langle \text{Definiție tip} \rangle ::= \langle \text{Identificator} \rangle = \langle \text{Tip} \rangle$
11.  $\langle \text{Variabile} \rangle ::= \text{var } \langle \text{Declarație variabile} \rangle ;$   
 $\quad \{ \langle \text{Declarație variabile} \rangle ; \}$
12.  $\langle \text{Declarație variabile} \rangle ::=$   
 $\quad \langle \text{Identificator} \rangle \{, \langle \text{Identificator} \rangle \} : \langle \text{Tip} \rangle$
13.  $\langle \text{Subprograme} \rangle ::= \{ \langle \text{Funcție} \rangle; \mid \langle \text{Procedură} \rangle; \}$
14.  $\langle \text{Tip} \rangle ::= \langle \text{Identificator} \rangle \mid \langle \text{Tip enumerare} \rangle \mid$   
 $\quad \langle \text{Tip subdomeniu} \rangle \mid \langle \text{Tip tablou} \rangle \mid$   
 $\quad \langle \text{Tip articol} \rangle \mid \langle \text{Tip mulțime} \rangle \mid$   
 $\quad \langle \text{Tip fișier} \rangle \mid \langle \text{Tip referință} \rangle$
15.  $\langle \text{Tip enumerare} \rangle ::= (\langle \text{Identificator} \rangle \{, \langle \text{Identificator} \rangle \})$
16.  $\langle \text{Tip subdomeniu} \rangle ::= \langle \text{Constantă} \rangle \dots \langle \text{Constantă} \rangle$
17.  $\langle \text{Tip tablou} \rangle ::= [ \text{packed} ] \text{array} ( . \langle \text{Tip} \rangle \{, \langle \text{Tip} \rangle \} . )$   
 $\quad \text{of } \langle \text{Tip} \rangle$
18.  $\langle \text{Tip mulțime} \rangle ::= [ \text{packed} ] \text{set of } \langle \text{Tip} \rangle$
19.  $\langle \text{Tip fișier} \rangle ::= [ \text{packed} ] \text{file of } \langle \text{Tip} \rangle$
20.  $\langle \text{Tip referință} \rangle ::= ^\langle \text{Tip} \rangle$
21.  $\langle \text{Tip articol} \rangle ::= [ \text{packed} ] \text{record } \langle \text{Listă câmpuri} \rangle [;] \text{end}$
22.  $\langle \text{Listă câmpuri} \rangle ::= \langle \text{Parte fixă} \rangle [; \langle \text{Parte variante} \rangle ] \mid$   
 $\quad \langle \text{Parte variante} \rangle$
23.  $\langle \text{Parte fixă} \rangle ::= \langle \text{Secțiune articol} \rangle \{ ; \langle \text{Secțiune articol} \rangle \}$
24.  $\langle \text{Secțiune articol} \rangle ::= \langle \text{Nume câmp} \rangle \{, \langle \text{Nume câmp} \rangle \} :$   
 $\quad \langle \text{Tip} \rangle$
25.  $\langle \text{Parte variante} \rangle ::= \text{case } [ \langle \text{Identificator} \rangle : ] \langle \text{Tip} \rangle$   
 $\quad \text{of } \langle \text{Variantă} \rangle \{ ; \langle \text{Variantă} \rangle \}$



26.  $\langle \text{Variantă} \rangle ::= \langle \text{Constantă} \rangle \{, \langle \text{Constantă} \rangle \} : ( [ \langle \text{Listă cîmpuri} \rangle ] [;] )$
27.  $\langle \text{Funcție} \rangle ::= \langle \text{Antet funcție} \rangle ; \langle \text{Corp} \rangle \mid \langle \text{Antet funcție} \rangle ; \langle \text{Directivă} \rangle$   
 $\mid \textbf{function} \langle \text{Identificator} \rangle ; \langle \text{Corp} \rangle$
28.  $\langle \text{Antet funcție} \rangle ::= \textbf{function} \langle \text{Identificator} \rangle$   
 $[ \langle \text{Listă parametri formali} \rangle ] : \langle \text{Identificator} \rangle$
29.  $\langle \text{Procedură} \rangle ::= \langle \text{Antet procedură} \rangle ; \langle \text{Corp} \rangle \mid$   
 $\langle \text{Antet procedură} \rangle ;$   
 $\langle \text{Directivă} \rangle \mid$   
 $\textbf{procedure} \langle \text{Identificator} \rangle ; \langle \text{Corp} \rangle$
30.  $\langle \text{Antet procedură} \rangle := \textbf{procedure} \langle \text{Identificator} \rangle$   
 $[ \langle \text{Listă parametri formali} \rangle ]$
31.  $\langle \text{Listă parametri formali} \rangle ::=$   
 $( \langle \text{Parametru formal} \rangle \{ ; \langle \text{Parametru formal} \rangle \} )$
32.  $\langle \text{Parametru formal} \rangle ::= [\textbf{var}] \langle \text{Identificator} \rangle$   
 $\{ , \langle \text{Identificator} \rangle \} : \langle \text{Identificator} \rangle \mid \langle \text{Antet funcție} \rangle \mid$   
 $\langle \text{Antet procedură} \rangle$
33.  $\langle \text{Instrucțiune} \rangle ::= [ \langle \text{Etichetă} \rangle : ] \langle \text{Instrucțiune neetichetată} \rangle$
34.  $\langle \text{Instrucțiune neetichetată} \rangle ::=$   
 $\langle \text{Atribuire} \rangle \mid \langle \text{Apel procedură} \rangle \mid \langle \text{Instrucțiune compusă} \rangle \mid$   
 $\langle \text{Instrucțiune if} \rangle \mid \langle \text{Instrucțiune case} \rangle \mid$   
 $\langle \text{Instrucțiune while} \rangle \mid \langle \text{Instrucțiune repeat} \rangle \mid$   
 $\langle \text{Instrucțiune for} \rangle \mid \langle \text{Instrucțiune with} \rangle \mid$   
 $\langle \text{Instrucțiune goto} \rangle \mid \langle \text{Instrucțiune de efect nul} \rangle$
35.  $\langle \text{Atribuire} \rangle ::= \langle \text{Variabilă} \rangle := \langle \text{Expresie} \rangle \mid$   
 $\langle \text{Nume funcție} \rangle := \langle \text{Expresie} \rangle$
36.  $\langle \text{Apel procedură} \rangle ::= \langle \text{Nume procedură} \rangle$   
 $[ \langle \text{Listă parametri actuali} \rangle \mid \langle \text{Listă parametri de ieșire} \rangle ]$
37.  $\langle \text{Listă parametri actuali} \rangle ::=$   
 $( \langle \text{Parametru actual} \rangle \{ , \langle \text{Parametru actual} \rangle \} )$
38.  $\langle \text{Parametru actual} \rangle ::= \langle \text{Expresie} \rangle \mid \langle \text{Variabilă} \rangle \mid$   
 $\langle \text{Nume funcție} \rangle \mid \langle \text{Nume procedură} \rangle$
39.  $\langle \text{Nume funcție} \rangle ::= \langle \text{Identificator} \rangle$
40.  $\langle \text{Nume procedură} \rangle ::= \langle \text{Identificator} \rangle$
41.  $\langle \text{Listă parametri de ieșire} \rangle ::=$   
 $( \langle \text{Parametru de ieșire} \rangle \{ , \langle \text{Parametru de ieșire} \rangle \} )$
42.  $\langle \text{Parametru de ieșire} \rangle ::=$   
 $\langle \text{Expresie} \rangle [ : \langle \text{Expresie} \rangle [ : \langle \text{Expresie} \rangle ] ]$
43.  $\langle \text{Instrucțiune compusă} \rangle ::= \textbf{begin} \langle \text{Instrucțiune} \rangle$   
 $\{ ; \langle \text{Instrucțiune} \rangle \} \textbf{end}$

44. *<Instrucțiune if> ::=*  
*if <Expresie booleană> then <Instrucțiune>*  
*[else <Instrucțiune>]*
  45. *<Instrucțiune case> ::=*  
*case <Expresie> of [<Caz> {; <Caz>}] [;] end*
  46. *<Caz> ::= <Constantă> {, <Constantă>} : <Instrucțiune>*
  47. *<Instrucțiune while> ::=*  
*while <Expresie booleană> do <Instrucțiune>*
  48. *<Instrucțiune repeat> ::= repeat <Instrucțiune>*  
*{;<Instrucțiune>}until <Expresie booleană>*
  49. *<Expresie booleană> ::= <Expresie>*
  50. *<Instrucțiune for> ::=*  
*for <Variabilă> ::= <Expresie> <Pas> <Expresie>*  
*do <Instrucțiune>*
  51. *<Pas> ::= to | downto*
  52. *<Instrucțiune with> ::= with <Variabilă>*  
*{, <Variabilă>} do <Instrucțiune>*
  53. *<Instrucțiune goto> ::= goto <Etichetă>*
  54. *<Instrucțiune de efect nul> ::=*
  55. *<Variabilă> ::= <Identificator> | <Variabilă> (. <Expresie>*  
*{ , <Expresie> } .) | <Variabilă>.*  
*<Nume câmp> | <Variabilă>*
  56. *<Nume câmp> ::= <Identificator>*
  57. *<Expresie> ::= <Expresie simplă>*  
*{<Operator relațional> <Expresie simplă>}*
  58. *<Operator relațional> ::= < | <= | = | >= | > | <> | in*
  59. *<Expresie simplă> ::= [+ | -] <Termen>*  
*{<Operator aditiv> <Termen>}*
  60. *<Operator aditiv> ::= + | - | or*
  61. *<Termen> ::= <Factor>*  
*{<Operator multiplicativ> <Factor>}*
  62. *<Operator multiplicativ> ::= \* | / | div | mod | and*
  63. *<Factor> ::= <Variabilă> | <Constantă fără semn> |*  
*<Apel funcție> | not <Factor> |*  
*(<Expresie>) | <Constructor mulțime>*
  64. *<Apel funcție> ::= <Nume funcție>*  
*[ <Listă parametri actuali> ]*
  65. *<Constantă fără semn> ::= <Număr fără semn> |*  
*<Șir de caractere> | <Identificator> | nil*
  66. *<Constructor mulțime> ::= (. [ <Specificare element>*  
*{ , <Specificare element> } ] .)*
  67. *<Specificare element> ::= <Expresie> [ .. <Expresie> ]*
- Notă.* Simbolurile neterminale [ și ] din formulele (55) și (66) sînt redată prin simbolurile echivalente respectiv ( și ).



## BIBLIOGRAFIE

- Munteanu F., Ionescu T., Muscă Gh. ș.a.*, Programarea calculatoarelor. Manual pentru liceele de informatică, clasele X-XII, Editura didactică și pedagogică, R. A., București, 1995.
- Boian F., Cioban V., Frențiu M. ș.a.*, Programarea PASCAL. Programe ilustrative și probleme propuse pentru elevi și studenți. Editura *Promedia plus computers*, Cluj-Napoca, 1995.
- Bălănescu T., Gavrilă Ș., Georgescu H. ș.a.*, Programarea în limbajele PASCAL și Turbo PASCAL, vol. 1, 2, Editura Tehnică, București, 1992.
- Rancea D.*, Limbajul Turbo PASCAL., vol. 1, 2, Editura *Libris*, Cluj, 1993.
- Sorin T.*, Tehnici de programare. Editura *L&S Informat*, București 1996.
- Sandor Kovacs*, Turbo PASCAL 6.0. Ghid de utilizare, Editura *MicroInformatica*, Cluj-Napoca, 1993.
- Cristea V., Dumitru P., Giumale C. ș.a.*, Dicționar de informatică. Editura Științifică și Enciclopedică, București, 1981.
- Йенсен К., Вирт Н. Паскаль*, Руководство пользователя, М., Editura *Финансы и статистика*, 1989.
- Вирт Н.*, Алгоритмы + Структуры данных = Программы, М., Editura *Мир*, 1985.
- Вирт Н.*, Алгоритмы и структуры данных, М., Editura *Мир*, 1989.